Models for design, implementation and deployment of 3D Collaborative Virtual Environments

Thierry Duval

▶ **To cite this version:**

Thierry Duval. Models for design, implementation and deployment of 3D Collaborative Virtual Environments. Graphics [cs.GR]. Université Rennes 1, 2012. tel-00764830

HAL Id: tel-00764830
https://theses.hal.science/tel-00764830

Submitted on 13 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Habilitation à Diriger des Recherches

présentée

devant l'Université de Rennes 1

pour obtenir

l'Habilitation à Diriger des Recherches
Mention Informatique

par

Thierry Duval
Équipe d'accueil : IRISA – VR4i
École Doctorale : Matisse
Composante universitaire : ISTIC

Titre:

Models for design, implementation and deployment of 3D Collaborative Virtual Environments

Soutenue le 28 November 2012 devant la commission d'examen

M. :	Jean-Marc	Jézéquel	Président
MM. :	Sabine	Coquillart	Rapporteurs
	Jean-Pierre	Jessel	
	Marc Erich	Latoschik	
MM. :	Joëlle	Coutaz	Examinateurs
	Pascal	Guitton	
	Jacques	Tisseau	

Contents

Introduction

3D Collaborative Virtual Environments

What is it?

A 3D Virtual Environment (3D VE) is a virtual environment where 3D objects are displayed to a user. A user of such an environment is involved in a perception/action loop [SSO94], and the success of his interactions contributes to his feeling of presence in the virtual environment [ZJ98]. Usually he can interact with this virtual environment through dedicated input devices. Chris Hand [Han97] proposes three categories of interactions: the navigation (the interaction with the viewpoint of the user), the manipulation of the virtual objects of the virtual environment (object selection, object manipulation), and the application control (interaction with 3D widgets in order to change some parameters of the virtual environment). This is very similar to the four categories proposed by Bowman et al. [BJH99] where interaction with the objects of the world is explicitly decomposed into selection and manipulation. Many efficient interaction techniques have been developed in this area in the past decade [BKLP04], and due to new 3D input devices and 3D displays becoming widely available for everyone, research in new 3D user interfaces is more relevant than ever [BCF+08].

A 3D Collaborative Virtual Environment (3D CVE) is an interactive 3D virtual environment where several local or distant users can join to share a collaborative interaction experience.

When an interactive 3D Virtual Environment is deployed upon an immersive display system, such as a CAVE™ [CNSD93], or a Head-Mounted Display (HMD), or a workbench, or simply a big screen, we can say that we are using Virtual Reality techniques in order to explore this 3D Virtual Environment and interact with it.

What is the use for them?

Object manipulation is one of the most fundamental tasks of 3D interaction in Virtual Reality (VR), and collaborative manipulation of virtual objects by multiple users is a very promising area [BGRP01].

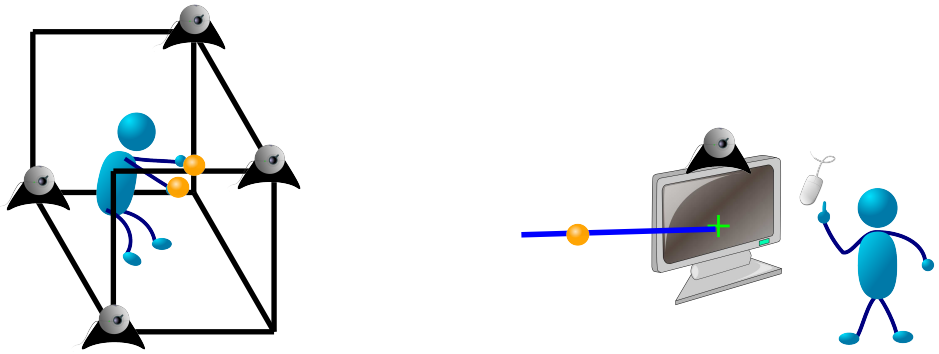
Collaborative manipulation of objects is indeed necessary in many different applications of VR such as virtual prototyping, training simulations or assembly and maintenance simulations [RSJ02]. In such virtual collaborative tasks, all the users should participate naturally and efficiently to the motion applied to the object manipulated in the VE [FN98, LGH98]. Another common use of 3D CVE is for virtual navigation: collaborative visits (museums, cultural heritage, architectural/urban project reviews), collaborative games (cars races).

3D CVE intend to make the users not just remotely communicate, but rather really interact together by sharing interactions in the 3D virtual environment. These interactions can happen on distinct objects, or on different parts of a same object, or even on the same part (at the same time) of a shared virtual object [MAP99].

Example of Design of a 3D Collaborative Virtual Environment

Today there are many kinds of virtual reality display devices that can be linked together in order to obtain collaborative VR applications. Nevertheless, designing these collaborative applications is still complex, especially if we want to allow the users to share highly interactive and immersive collaborative experiences.

For example, let's consider the design of a collaborative application where two distant users are going to share a co-manipulation of a virtual table through a 3-point manipulation technique. We will suppose that one user will use a big cave-like system and that his head and his two hands will be tracked so that he will be able to use his two hands to drive 3D cursors (see Figure 1 (a)), while the other user will use a desktop system and a 2D input device (such as a 2D mouse) for driving a 3D ray and that his head will be tracked so that his point of view will change when he will move in front of his screen (see Figure 1 (b)).



(a) (b)

Figure 1: (a) The first user in his cave-like system — (b) The second user with his desktop system

Indeed, the designer of such a distributed and multi-user application is facing several complex topics to meet its requirements:

1. at an abstract level, he has to describe both the interactive content of the virtual universe and the collaborative interaction tools that will be used to drive this content, through dedicated new collaboration metaphors able to give to the users the best collaborative 3D interaction experience. Here, there will be a table, two 3D cursors, a 3D ray, an avatar of the position of the head of the first user, and an avatar of the position of the head of the second user. The 3D cursors and the 3D ray will be able to manipulate the table by using a 3-point manipulation technique.

2. at a network level, he has to choose a model for the communications between distant users: a distribution mode for the virtual objects that will be part of the shared virtual environment, and also a synchronization mode for distant sites and machines. Here, he could decide that the table will be on a server, that the two 3D cursors and the head of the first user will be on the first user's machine, and that the 3D ray and the head of the second user will be on the second user's machine. He can also decide that there will be a strong synchronization between these three machines.

3. at a presentation level, he has to adapt the graphic rendering to the hardware output devices and he also has to make a link between the abstract interaction tools and the input devices that will be used at run-time to drive them. Here, for the first user he will have to use a 3D graphics API able to manage a stereo rendering on several displays with head-tracking,

and explain that his two 3D cursors will be driven by the position provided by two targets of his tracking system, while for the second user he will have to use a 3D graphics API for rendering on a simple screen with a low-cost tracking system to track his head to deform the field of view, and explain that his 3D ray will be driven by a 2D mouse (for example using the wheel of the mouse in order to provide the depth).

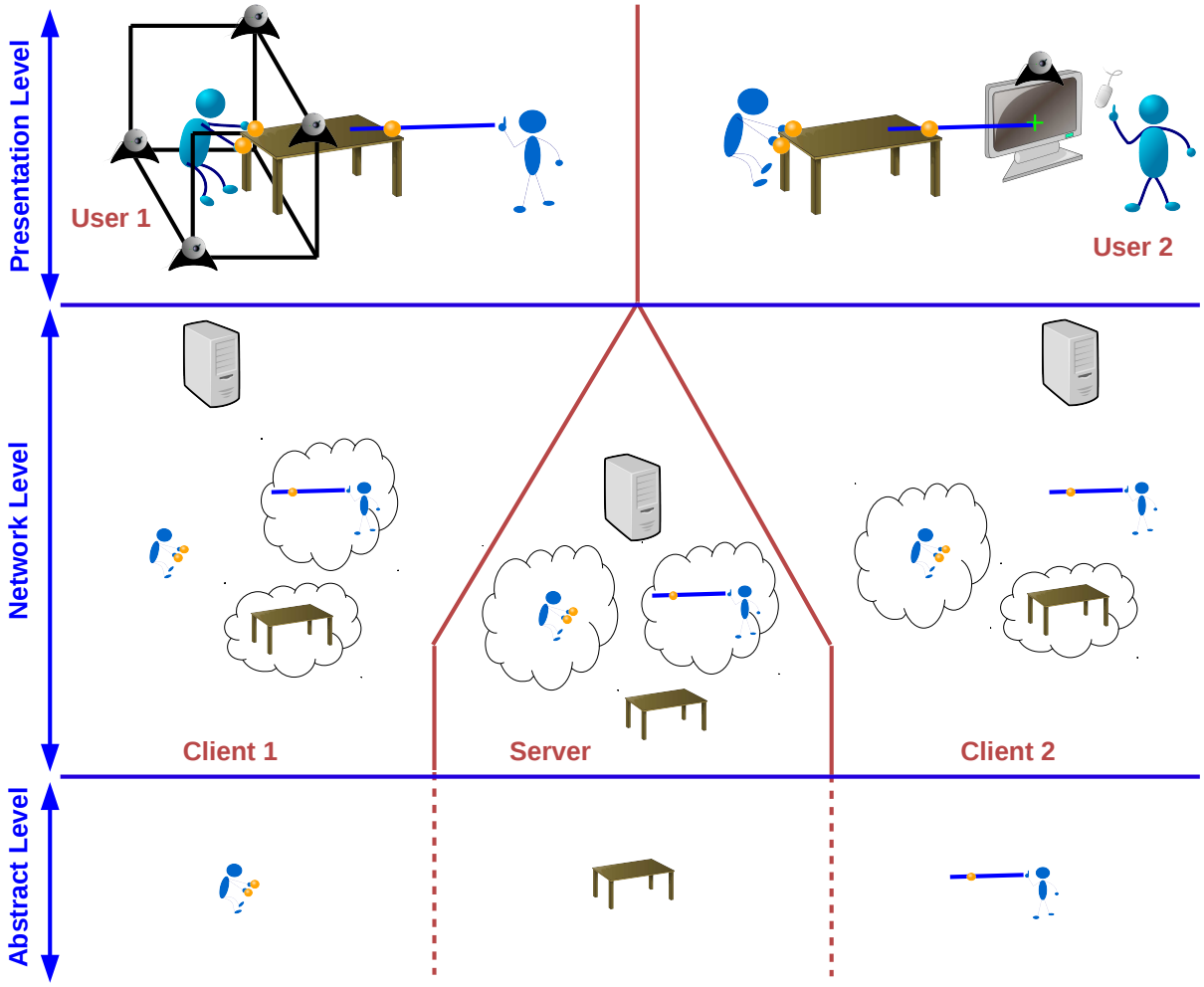


Figure 2: The different levels involved in a global CVE architecture

The different levels involved in the global architecture of such a CVE are illustrated in Figure 2.

As the hardware and software features of such a system may change over the time, the designer of such a CVE should take into account some more points:

1. at a network level, he should be able to change the distribution characteristics of the virtual objects and the synchronization mode of the system, and to let the users join and leave dynamically the collaborative session.

2. at the connection between the abstract level and the presentation level, he has to make the virtual environment as independent as possible from the input and output devices used at run-time, to be able to adapt it dynamically (at run-time if necessary) to various kinds of hardware (tracking systems, input devices, displays) and software (3D graphics API).

3. at the connection between the abstract level and the network level, he has also to make the virtual environment as independent as possible from these network features.

Last, still at the abstract level, he also should be able to integrate a representation of the run-time hardware components in the virtual environment (to make a link between the different levels), to make the users aware of the limitations of these components. It leads to the creation of new virtual objects that must be added to the CVE: a representation of the bounds of the tracking systems of each user, a representation of the display surfaces of the first user, and a representation of the field of view of the second user.

The resulting global architecture of the CVE with this extended content is illustrated in Figure 3. It shows that the designer of such a CVE will have a quite difficult exercise to realize as he will have to embed into the shared virtual environment a description of the physical workspace of the users (known only at the presentation level) while he will have to maintain a strong independence of the abstract level from the presentation level.

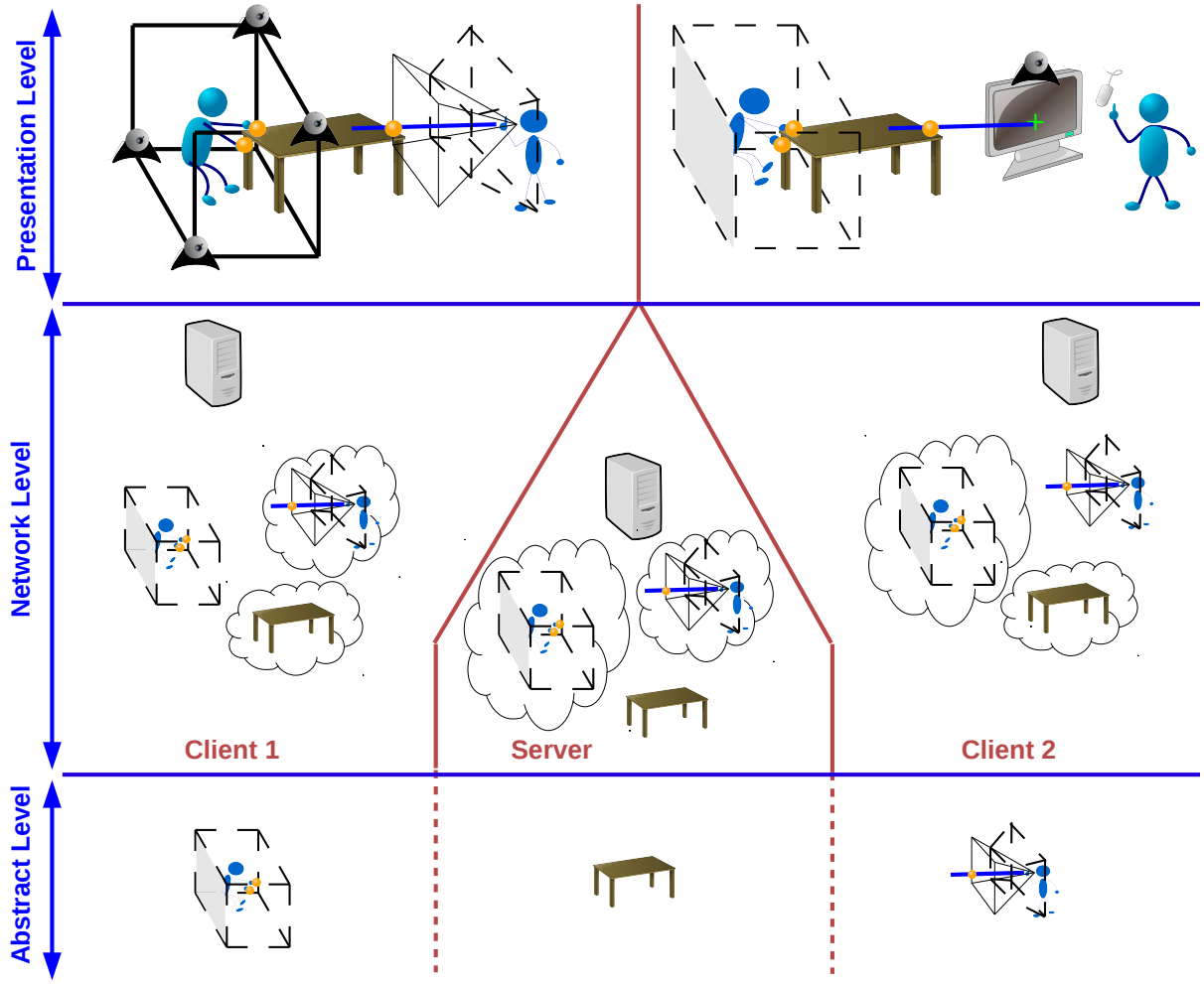


Figure 3: The different levels involved in a revisited global CVE architecture

Main Steps of the Design of 3D Collaborative Virtual Environments

From this example, we can extract some main steps in the global design process of a CVE.

1 – Choosing a model for distribution and synchronization

This task consists in determining the best way, according to the needs of the users and to the constraints of the system, to distribute and synchronize each shared object of the CVE, in order to have the best possible consistency of the CVE between all the machines involved at run-time.

The designer of a CVE will have to answer many questions such as: should all the shared objects be centralized on a server, or totally replicated on each site, or spread over the sites with a hybrid distribution model? How should distant sites be synchronized: with a strong synchronization or with a relaxed one? Can we tolerate and manage some temporary relaxation of the synchronization due to network breakdowns? Should we allow some shared objects to migrate individually from one site to another to provide load-balancing or to make a shared object closer to its user?

Some of these choices may be imposed by some requirements of the collaborative application or by the features provided by the framework used to implement the CVE.

2 – Adapting the Virtual Environment to various hardware systems

This task consists in adapting a VR application to the software and to the hardware input and output devices that are available at run-time.

It can be difficult to address different kinds of hardware devices with the same software. For example, some 3D graphics API are adequate for a rendering using a single screen but not for a rendering in an immersive system (with head-tracking and stereo-vision) or on a mobile device. Similarly, it should be possible to deploy a VR application with different kinds of input devices, according to what is available at run-time: sophisticated input devices such as ART flysticks, low-cost devices such as the Nitendo Wiimote or the Microsoft Kinect, or even simply a 2D mouse.

So, designers of VR applications need software architectural models in order to clearly separate the core of the VR application from the software used for the rendering and for the interactions.

3 – Designing interaction and collaboration in the VE

The description of a virtual environment is often limited to the 3D geometrical features of its virtual objects. This task is essential, but it takes into account neither the interactive and collaborative features of the virtual objects, nor their behavior. Furthermore, in a collaborative context, the interaction capabilities of a virtual object can vary accordingly to the user who interacts.

The designer of an interactive CVE must also describe in which way each interactive object will make users aware of its interactive potential: how will it propose interaction to a user? How will it make the user understand its constraints? How will it make users aware of the collaboration possibilities? This designer will also have to define the software interaction tools that will be used for these interactions. This part should complete the description of the CVE.

Last, most of the time single-user interaction tools and metaphors are not adapted to offer efficient collaboration between users of a CVE. Some of these tools and metaphors can be adapted for collaborative interactions, and new really collaborative metaphors need to be proposed to enhance real multi-user collaborative interactions, with dedicated collaborative feedback.

4 – Embedding the users' physical workspaces within the CVE

It is also quite important to take into account the features of the hardware devices available at run time, because they have a strong impact on user immersion and interaction. For example the knowledge of the boundaries of an immersive display can prevent the user from disrupting the feeling of immersion by colliding the display. In a collaborative context, it is also interesting to make users aware of the interaction capabilities of the other users. So, designers will have to find a model for the features of the hardware devices used at run-time in order to embed them in the virtual environment as a new kind of virtual objects.

Addressing the essential requirements about the design of 3D CVE

We have identified six topics that must be addressed when designing a CVE to meet most of their requirements. These topics focus either on the global architecture of the CVE, at a system or at a software level, or on the details of the collaborative interactions, at the object level.

For each of these identified topics, we will present a state of the art about the solutions that can address this topic, then we will show our own contributions: how we improve existing solutions and what are our new propositions.

Part I: System and Software Architectural Models for Collaborative Virtual Environments

Chapter 1 (System Architectures for Data Distribution in Collaborative Virtual Environments) makes a point about pros and cons of the main CVE system architectures. Our main contribution to this topic is to propose to allow CVE designers to mix in a same CVE the three main distribution models usually encountered (centralized on a server, totally replicated on each site, or distributed), and to offer an implementation of this concept in the Collaviz framework.

Chapter 2 (Synchronization Models for Collaborative Virtual Environments) deals with synchronization between distant sites. Our contribution to this point is about adaptation to the jitter of the network latency, with several synchronization groups of users, some migration mechanisms for virtual objects, and some metaphors dedicated to make the users aware of what is occurring on the network.

Chapter 3 (Software Architectural Models for 3D CVE) explains why it is so important to make a strong separation between the core of the virtual environment and the software and hardware features used to implement and deploy the virtual environment. We present a state of the art about software architectural models for interactive and collaborative software, and we propose our own solution, the PAC-C3D model, which is able to deal with the three main distribution modes encountered in CVE.

Part II: Models for Designing Collaborative Interactions

Chapter 4 (Modeling Interaction and Collaboration) presents how to go one step beyond geometric modeling, by adding interactive and collaborative features to the model of the virtual objects. Here our main contribution is about a unified model of dialog between interactive objects and interaction tools. We have also proposed an extension to Collada in order to describe interactive and collaborative properties of these interactive objects and interaction tools.

Chapter 5 (Metaphors for Collaborative Interactions) talks about metaphors for interaction within CVE. It shows that usual 3D interaction metaphors must generally be adapted to fit with collaborative interactions, and to make users aware of this collaboration. We have contributed to this field through new metaphors for multi-user interaction, with new interaction tools acting at the object level.

Chapter 6 (Modeling Users' Physical Workspaces) shows that we need to take into account the users' physical environment at run-time in order to adapt the CVE to the hardware input and output devices of the users. This is why we propose the Immersive Interactive Virtual Cabin (IIVC) concept to embed in the CVE a 3D model of the users' physical workspaces.

Part I

System and Software Architectural Models for CVE

Collaborative Virtual Environments (CVE) enable users to collaborate and interact together by sharing a common virtual environment. Ensuring the global consistency of such a virtual environment is very important to provide efficient collaboration between users. However, users sharing a CVE may be scattered over different physical locations, so CVE systems have to guarantee the virtual environment consistency despite network issues such as low bandwidth or network latency. Absolute consistency is nearly impossible to achieve because it would prejudice the responsiveness of the system during user interactions. So, CVE systems have to deal with a trade-off between consistency and responsiveness of the system. This is a network level viewpoint upon a CVE global architecture, as illustrated in Figure I.1.

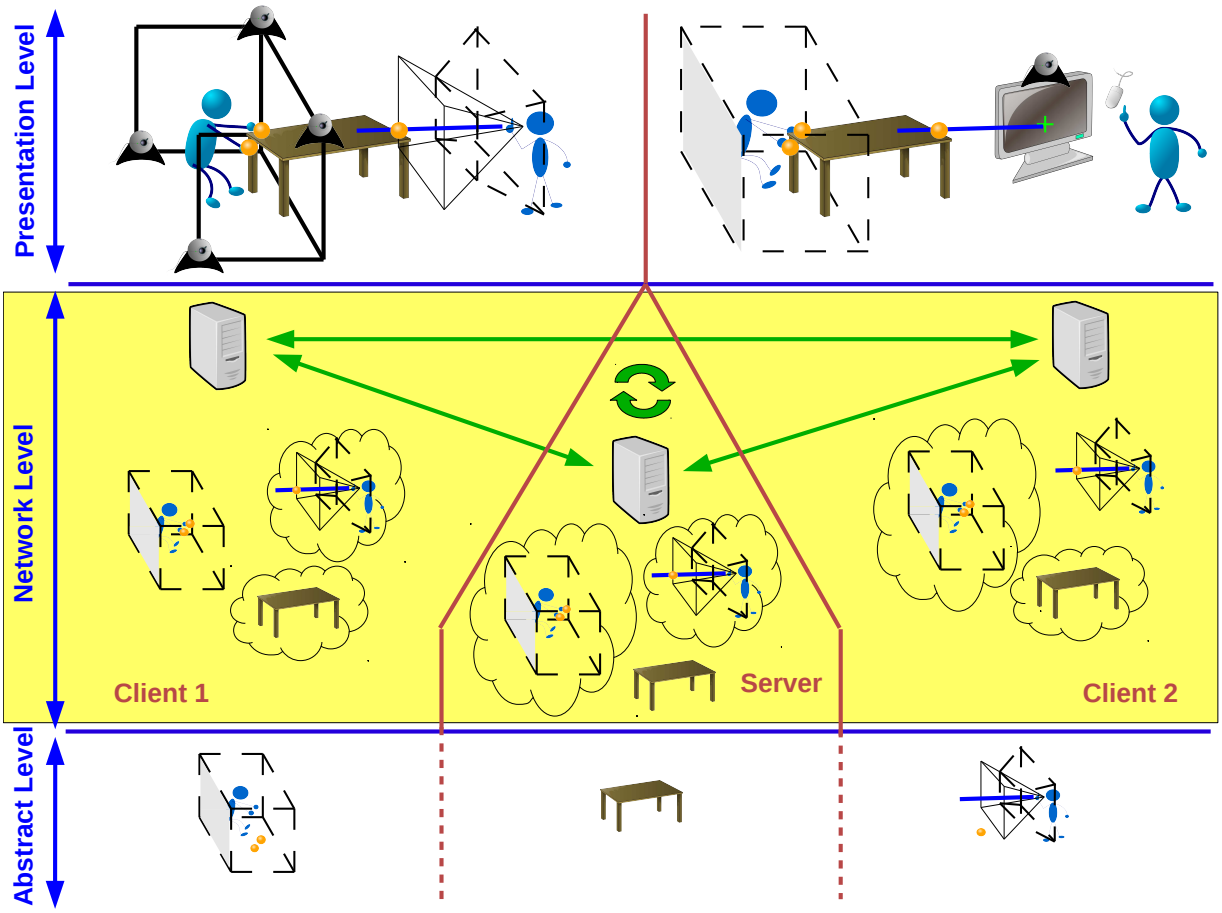


Figure I.1: Modeling distribution and synchronization in a CVE

In chapter 1 (System Architectures for Collaborative Virtual Environments) we present a survey of architectures and mechanisms used to improve the consistency of a shared virtual environment. Architectures of CVE systems are studied according to their impact on consistency. We identify three main different distribution modes, each one has some advantages upon the two others. So our contribution to this topic is a new framework that implements these three distribution modes. The choice of the distribution mode can be made individually for each virtual object of the world, according to its requirements, and this distribution mode can be modified at run-time.

In chapter 2 (Synchronization Models for Collaborative Virtual Environments) consistency maintenance mechanisms are also examined. First, a time synchronization must be achieved in order to enable users to perceive the same state of the virtual environment at the same time. Second, the virtual environment can be seen as a database shared by several users, so CVE systems must manage users' concurrent access to shared virtual objects. Our conclusion about synchro-

nization is that usual synchronization modes can be improved to deal with network latency. Our contributions are about management and awareness of the jitter of the network latency. We propose to dynamically organize, at run-time, several synchronization groups, and to make the users aware of the latency between some sites of a CVE.

When designing a VR application, one can choose between the many hardware devices and software API that are available for managing the interactions of the users and the rendering of the virtual environment. Typically, if these software and hardware devices are too closely-coupled to the core representation of the virtual environment, it will be very difficult to adapt the VR application to other kinds of devices (many input devices can be used to provide similar interactions) or to other kinds of rendering software. So we need software architectural models to make an efficient link between the different levels of a CVE, as illustrated in Figure I.2.

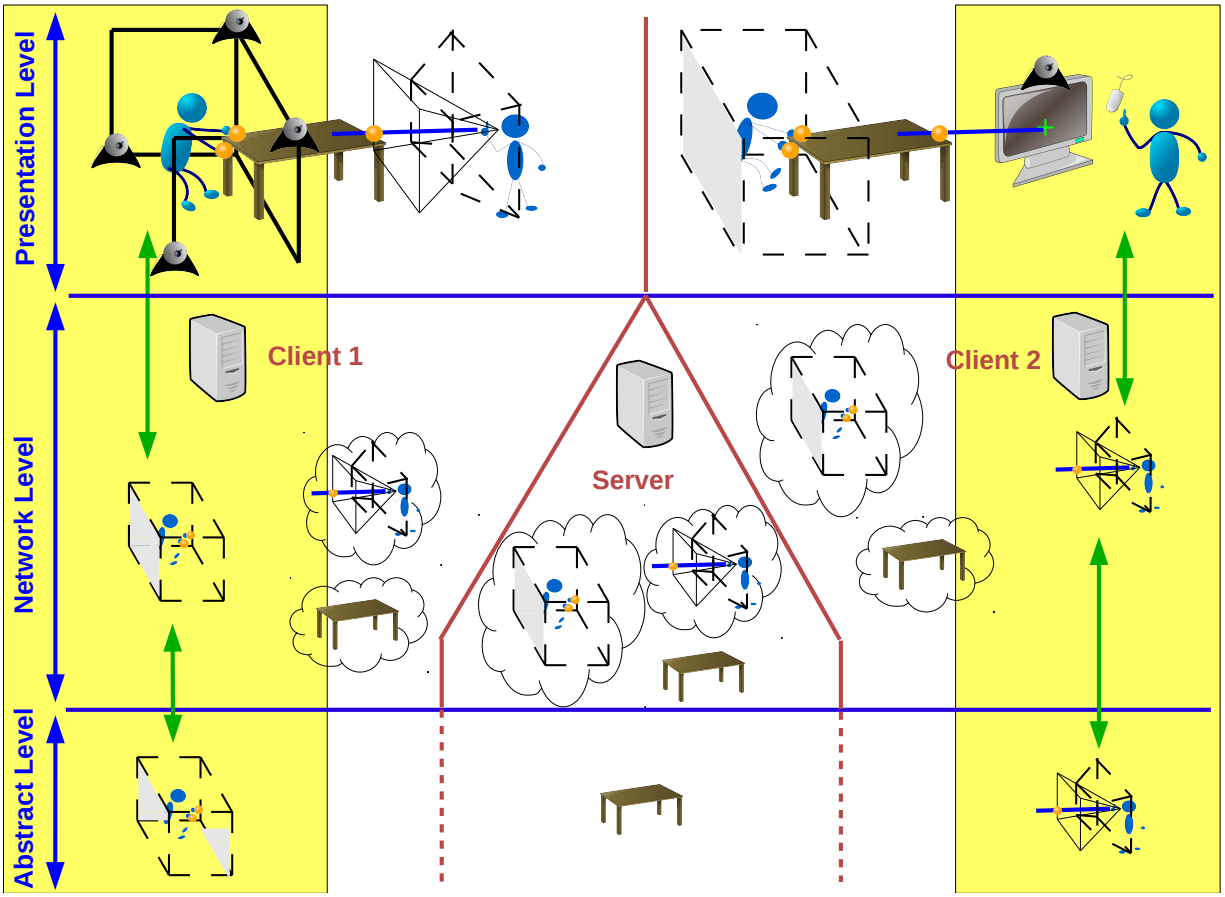


Figure I.2: Making the link between the different levels of a CVE architecture

In chapter 3 (Software Architectural Models for 3D CVE) we propose PAC-C3D as a new software model for 3D CVE. This model merges the results from two research fields: distribution models for CVE and HCI design for computer-supported cooperative work. PAC-C3D uses explicit interfaces to ensure a strong separation between the core functions of each object of a virtual environment, its (visual) representations, and its collaborative features such as synchronization and consistency maintenance between remote users. PAC-C3D makes it possible to design a CVE with low dependency between the core functions, the distribution mode and the 3D graphics API used for each virtual object. It explicitly deals with the main distribution modes encountered in CVE and it makes it easy to use different 3D graphics API for different nodes involved in the same collaborative session, providing interoperability between these 3D graphics API. It also makes it possible to integrate other kinds of 3D representations such as physics engines into the CVE.

Chapter 1

System Architectures for Collaborative Virtual Environments

1.1 Introduction

One of the main goals of Collaborative Virtual Environments (CVE) is to enable users to work together in a natural way and to provide them a truly interactive experience. Generally, each user uses his own computer to have individual interaction capabilities or to meet the others if they are not located at the same geographical place. So, this collaborative work must be achieved over a local area network (LAN) or a wide area network (WAN) between the users' computers that we will call "nodes". For example, in the case of the French ANR project Collaviz[1], remote experts have to analyze together scientific data using an Internet connection through a secured proxy. Such network connections have a strong impact on the consistency of the shared virtual environment because they delay the transmission of the virtual environment changes. While some users of a CVE can interact through immersive devices, powerful computers, and high-bandwidth network connections, some other users can share the same CVE through simple workstations and low-bandwidth network connections. Even if not powerful enough computers or low-bandwidth network connections put some users at a disadvantage, these users have the right to share the same state of the virtual environment than the other users. Ensuring the CVE consistency is the best way to make possible an efficient collaboration between users because it can avoid conflicts between several user actions or misunderstandings when users perform collaborative tasks.

To define the consistency of a CVE, Delaney et al. [DWM06a] explain that a CVE must be considered as a distributed database with users modifying it in real-time. Consistency maintenance consists in ensuring that this database is the same for all the users, i.e. users observe or interact with the same data. The consistency of a CVE can be characterized by the following three criteria:

- **synchronization:** (1) time synchronization: an event (state modification of a shared virtual environment) should happen simultaneously on all the nodes. (2) spatial synchronization: CVE objects should have the same location at the same time on all the nodes.
- **causality:** events order must be the same for all the users.
- **concurrency:** conflicts can occur when users change the same parameter of a virtual object at the same time. These conflicts have to be managed to avoid that users make their own modifications and have inconsistent states of the CVE.

Consistency is directly linked to system responsiveness. Responsiveness can be defined as the time needed by the system to respond to user actions. Responsiveness during interactions can be

[1] www.collaviz.org

quantified by the system latency, i.e. the time between a user action and the system response. This latency is often due to the transmission time over the network and to the processing delays of the events. Improving the consistency of a CVE can increase latency during interactions and vice versa. So, CVE systems must reach a trade-off between consistency and system responsiveness.

Latency is especially a problem for highly interactive applications. Delaney et al. [DWM06a] present several values of latency found in the literature. It seems that no consensus has been found about these values. The maximum latency values fluctuate between 40 and 300 ms to ensure real-time interactions, and a maximum latency of 100 ms seems sufficient for most of the applications. These values also depend on the jitter (the variation of the latency) of the system. Roberts et al. [RRS98] explain that jitter has a more significant impact on user performance than latency. It would be better to have a quite high and constant latency (i.e. with a low jitter), rather than a lower latency with a higher jitter.

In this chapter, section 1.2 describes the different architectures of CVE systems and their impact on consistency and system responsiveness. Then section 1.3 proposes a new solution to maintain the consistency in CVE systems.

1.2 Related work

Consistency and performance of a CVE are highly correlated with its system architecture. For example, some architectures maintain a strong consistency of the CVE but introduce latency during interactions. In contrast, other architectures accept a few inconsistencies but offer a better responsiveness during interactions. Previous state of the art reports classify CVE systems according either to how the nodes are connected together [DWM06b, JDGMT04] or to how data are distributed on these nodes [MZ97, Zam05]. Although these two characteristics are often interrelated, we chose to examine both of them independently for our classification: the kind of network architecture (section 1.2.1) and the kind of data distribution (section 1.2.2). This classification choice is motivated by the fact that more and more hybrid solutions mix different architectural choices to meet their requirements.

1.2.1 Network architectures

A CVE system is usually made up of several interconnected nodes that can be geographically scattered. Each node can communicate with the others through three main data transmission methods:

- *unicast:* transmission from one node to another one.
- *broadcast:* transmission from one node to all the others.
- *multicast:* transmission from one node to a subset of other nodes.

Delaney et al. [DWM06b] distinguish two basic network organizations used for CVE systems: the peer-to-peer architecture and the client/server architecture. They also introduce hybrid architectures that combine these two solutions.

1.2.1.1 Peer-to-peer architecture

The peer-to-peer architecture enables fast communications between pairs of users, because events are transmitted directly from one node to another one (see Figure 1.1). So, it enables a few users to have strong synchronization, and consequently closely coupled interactions. However, increasing the number of users will increase hugely the number of messages transmitted on the network, especially when an *unicast* transmission is used (if the session contains N members, a node has to send N-1 messages to transmit one event). Consequently, it is difficult to contact all the nodes

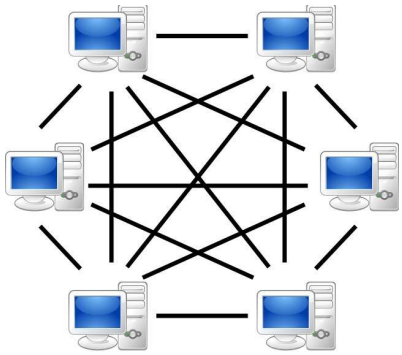


Figure 1.1: Peer-to-peer architecture.

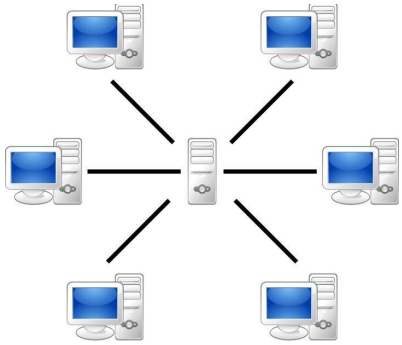


Figure 1.2: Client/server architecture.

at the same time to transmit virtual environment changes. So, time synchronization and global consistency of the CVE may be difficult to ensure. This kind of architecture appeared with the first CVE systems (Reality Build for Two [BBH+90] that connects only two computers, MR Toolkit [SG93]) and is used in military applications (SIMNET [CDG+93], NPSNET [MZP+94]).

Our first contribution to collaborative systems is the ARéVi distributed virtual reality toolkit [DMR+97, RHM+98]. Its kernel (a set of C++ classes) uses a peer-to-peer architecture to make it possible for several "applications" (what we call "ARéVi sessions") to share a common 3D universe. At first it was using broadcast for communication between sessions on a local area network, and unicast for communication between sessions on a wide area network.

1.2.1.2 Client/server architecture

This kind of architecture is based on a central server. All the nodes get connected to this server (see Figure 1.2). The central server manages the communication between different nodes and stores data. This kind of architecture enables the server to contact all the nodes at the same time. So, time synchronization and CVE consistency are easier to maintain than with the peer-to-peer architecture. However, when two users want to interact together, all the communications have to pass through the server, which increases latency during interactions. When the number of users increases, a bottleneck can occur on the server due to too many communication requests, so all the communications can be slowed down. For example, a client-server architecture is used in RING [Fun95], BrickNet [SSP+95], and ShareX3D [JFM+08].

1.2.1.3 Hybrid architecture

To overcome the previous limitations, some systems propose an hybrid network architecture that uses both peer-to-peer connections and one or several servers. For example, it is possible to speed up communications between nodes by using peer-to-peer connections, and it is possible to maintain a better consistency with a server.

In SPLINE [WAB+97], several servers share up-to-date information (messages, events, etc.) with peer-to-peer connections between these servers (see Figure 1.3). At the beginning of a session, the session manager connects nodes to one of these servers, then nodes only communicate with their assigned server. This solution avoids the bottleneck on the server when the number of users increases, and it makes it possible to easily connect nodes with slower or secured connections. Indeed, each server can perform additional processing such as compression or communication with a specific protocol. However, the use of too many servers can increase the system latency and the load of the servers.

Anthes et al. [AHV04] suggest another hybrid architecture to facilitate collaboration between nearby users (according to their location in the virtual environment) by reducing the latency be-

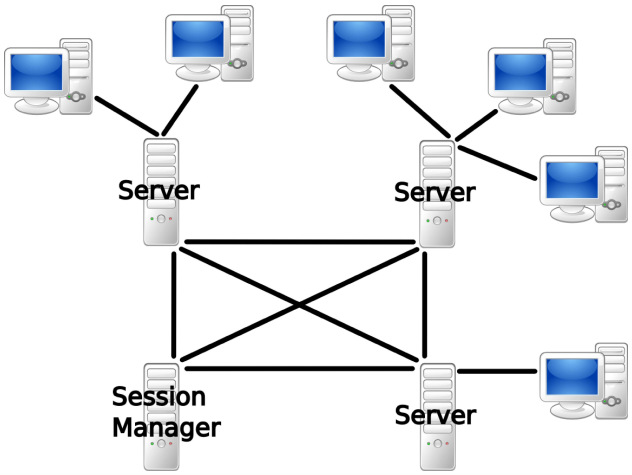


Figure 1.3: Several servers using peer-to-peer connections.

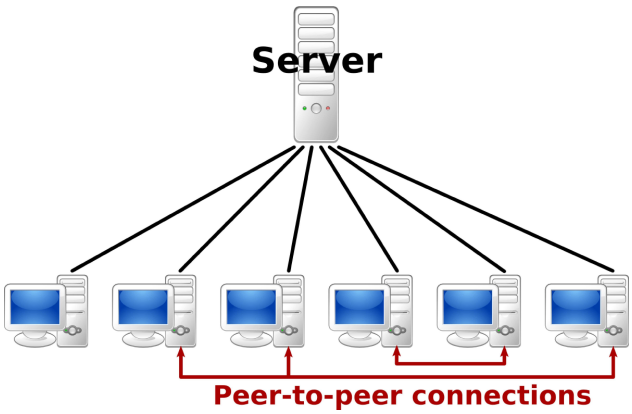


Figure 1.4: Temporary peer-to-peer connections between close users.

tween them. Users are connected to the CVE through a server. When some users come closer together in the virtual environment, temporary peer-to-peer connections are established between these users to increase the virtual environment consistency (see Figure 1.4).

CAVERN [LJD97] is based on a generic interface that defines the network connection and the data storage. Each node implements this interface to make possible a communication with all the other nodes. This node can act as a client or as a server, so any kind of network architectures can be implemented with this framework.

Our second contribution to distributed VR is the hybrid architecture of the OpenMASK platform [MAC+02], previously known as the GASP platform [DM00b, DM00a]. It uses peer-to-peer connections to send updates and events to nodes. It also uses a server to manage the identification of virtual objects and to dynamically add nodes during a session.

1.2.2 Models for data distribution

As stated by Macedonia et al. [MZ97], the location of the virtual environment data (i.e. geometric data, textures, etc.) is a critical decision when designing a CVE system. It determines which computers store this data and which computers execute the processing related to each virtual object. We distinguish three data distribution modes: centralized, homogeneously replicated (duplicated), and partially replicated (distributed). Further details about the impact of system architectures on CVE consistency can be founded in [FDGA10b].

1.2.2.1 Shared centralized world

Some systems such as Vistel [YTAK95] use one database shared by all the nodes. All the data relative to the virtual environment are stored on a central server. Similarly, behaviors of the CVE objects are executed on this server (see Figure 1.2.2.1). Users log on the server to join a session (it requires a client/server network architecture). When a user wants to modify an object, his node sends a request to the server. The server processes the modification request, then transmits the up-to-date state of the object to all nodes, including the one that asked for this modification (see Figure 1.2.2.1).

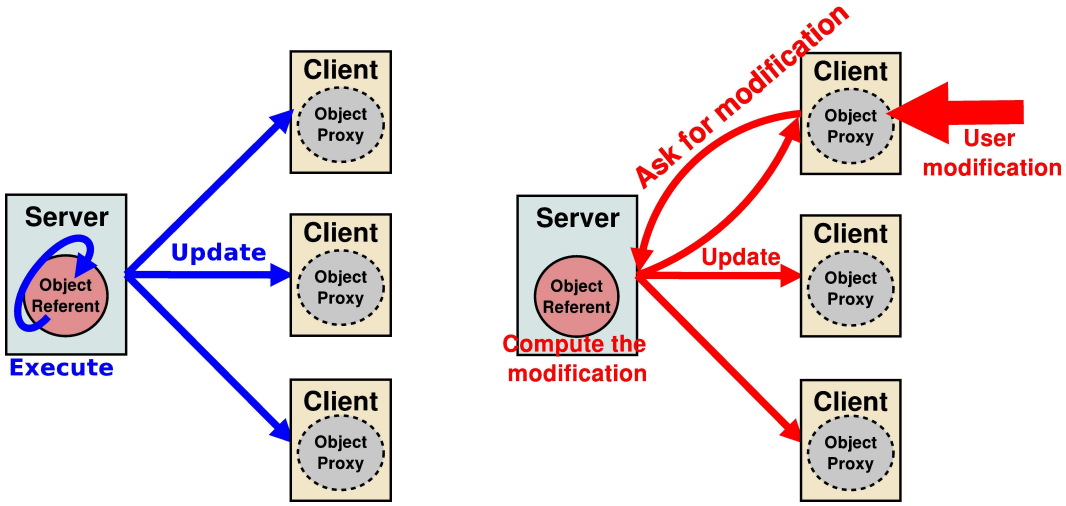


Figure 1.5: (a) Executions of object behaviors and (b) object modifications in a shared centralized world.

This method ensures consistency between all the nodes and avoids data replication, but it has two main drawbacks:

- During interactions, latency can increase when transmission delays occur between the clients and the server. Indeed, each modification request must pass through the server before returning to the user (see Figure 1.2.2.1). Users can get annoyed with this lack of responsiveness during interactions.
- With many users, a bottleneck can appear on the server because it has to send updates to all the nodes at the same time (especially with *unicast* connections).

1.2.2.2 Homogeneous replicated world

This kind of data distribution is used in many CVE systems (SIMNET [CDG+93], MR Toolkit [SG93]). All the nodes are initialized with the same database that contains all the information about the virtual environment (terrain, geometric models, textures, object behaviors, etc.). Similarly, bcDSG [NLSG03] replicate a shared scene graph on all the nodes. Data can already be present on the node when the user logs in (such as in most of the video games). Otherwise, data must be replicated from a server, or from the other nodes already connected to the session. During the session, the database evolves independently on each node and all object behaviors are executed locally (see Figure 1.2.2.2). Additionally, a synchronization mechanism can be used to control executions of object behaviors on each node. To maintain consistency, only object modifications and some special events (collision between two objects, etc.) are transmitted on the network in order to enable all nodes to update their database (see Figure 1.2.2.2).

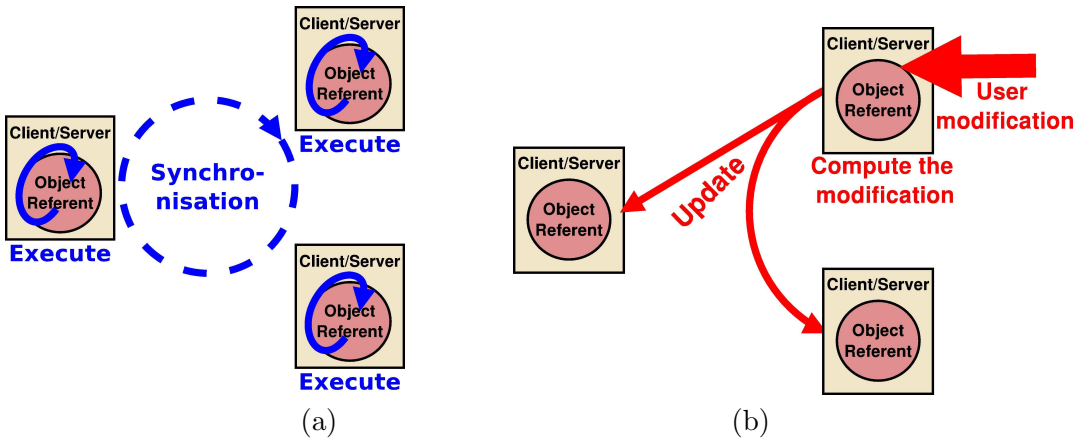


Figure 1.6: (a) Executions of object behaviors and (b) object modifications in an homogeneous replicated world.

This data distribution has two main advantages:

- The number and the size of messages transmitted on the network are really small because only update messages are sent.
- The latency is very low during user interactions. In fact, virtual object modifications are performed locally before being sent to the other nodes by using update messages.

However, data replication also has drawbacks:

- Data replication can introduce inconsistencies between users' virtual environments because of delays or losses during the transmission of update messages.
- Additional mechanisms must be provided to manage the concurrent access on each node. Indeed, a user can perform a local modification of an object, but modification conflicts can only be checked when the changes are transmitted to the other nodes.
- This solution is not really appropriate for very large data sets such as scientific data or complex CAD models, because each node may manage its own large database.
- This approach is not really flexible, especially if users want to add new objects in the initial database.

1.2.2.3 Partially replicated world (or distributed world)

Many CVE systems choose hybrid solutions between totally centralized and totally replicated data distributions in order to avoid the drawbacks of these two solutions. These hybrid solutions distribute the data and their processing among the nodes. It reduces the necessary resources and eases the consistency maintenance. For example, RAVE [GAW09] uses a central server to process object behaviors, and uses asynchronous transmission to perform "best effort" collaboration. Nodes maintain a local copy of the virtual environment, only receiving update messages from the server. In the literature, these hybrid solutions are called partially replicated world or distributed world.

To avoid the duplication of all the data on each node, data can be distributed on the network among these nodes. So the database can be seen as a shared and distributed memory. In DIVE [FS98], when a new user joins a collaborative session, only the necessary objects are replicated on his node instead of downloading the whole data of the virtual environment (see Figure 1.7). However, if this user needs other objects during the session, they must be dynamically downloaded.

DIVE uses peer-to-peer connections to manage communication between nodes (transmissions of messages or object data when needed). DIVE also uses a server to backup the data distributed

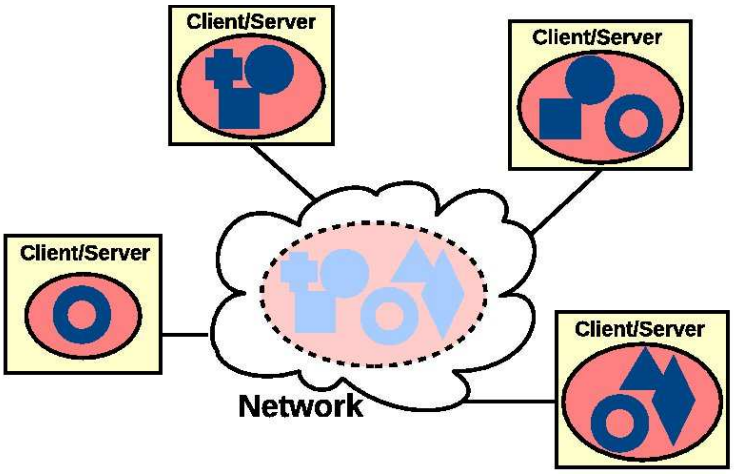


Figure 1.7: Data partially replicated on each node.

all over the nodes, in order to save the state of the virtual environment when a user logs out, and to restore this session later. This method makes it possible to share CVE between many users and to share a very large amount of data without necessarily duplicate this data on each node. However, dynamic transmission of data and consistency maintenance induce a high cost of communication between nodes. When an object changes, update messages must be sent to all the nodes even if they do not own this object. Indeed, a node cannot know which node owns the modified object.

The main difficulty of this partially replicated world is to dynamically download data without disturbing user interactions. According to Lee et al. [LLHL07], two solutions can be used:

- The prioritized transfer: this technique consists in selecting first the objects that are in the user field of view, and transferring these objects using level of details (LODs) or multi-resolution techniques. This solution maximizes graphical fidelity of the virtual environment as well as interactive performance by mediating graphical details and transmission overhead of the objects.

- Caching and prefetching the data that users will probably need: it makes data immediately available when users ask for it. However, this solution must predict efficiently which objects will interest first a user to define a loading priority. Generally, the distance between users and objects is used to determine this priority, assuming that users will be interested first by the closest objects. Other solutions suggest to add the moving direction of users or to use the objects type to determine the users' scope of interest. However, this prediction is difficult to achieve with lots of objects.

To reduce the cost of communications for the updates, BrickNet [SSP+95] uses a server to keep information about the shared objects: access rights, ownership, etc. It uses a client/server architecture: the server manages communication between nodes. Object behaviors are executed on each node, and the server keeps a list of nodes that share this object. When a user wants to modify an object, his node must first ask the server for the modification rights on this object (only one user can modify an object at the same time). When its request is granted, it can modify locally this object. Then the new state of the object is transmitted to the node that owns this object, using a list of object owners on the server. With this approach, the server makes it possible to ease the consistency maintenance and to manage concurrent access to objects.

In OpenMASK [MAC+02], data of the virtual environment are replicated on each node. Each object behavior is executed only on one node. To achieve this, OpenMASK uses a referent/proxies paradigm [DM00b, DM00a]. The referent is assigned to a particular node and defines the object

behavior. On the other nodes, proxies are created to represent the object. A proxy provides the same interface as the remote referent (same inputs, same outputs, etc.). However, there is no processing done locally and the proxy is driven by its referent (see Figure 1.8(a)). The OpenMASK kernel maintains the consistency between the referent and its proxies. When a user manipulates an object with the referent on his node, first the object is modified locally, second an update is sent to all the other nodes (see Figure 1.8(b)). If the referent of the manipulated object is not on his node, the object modification is computed first on the remote node that owns the referent. Then this node transmits updates to all the nodes, including the node that asked for the modification (see Figure 1.8(c)). In this second scenario, transmission delays on the network can introduce latency during interactions. However, this solution makes it possible: (1) to combine the processing power of all the nodes, (2) to ensure a better consistency of the virtual environment, and (3) to manage implicitly the concurrent access to the objects (in a first step, only the referent can be modified).

ARéVI [DMR+97] uses the same referent/proxies paradigm than OpenMASK except that it uses a dead-reckoning algorithm with simplified behavior models for predicting the state of the proxies.

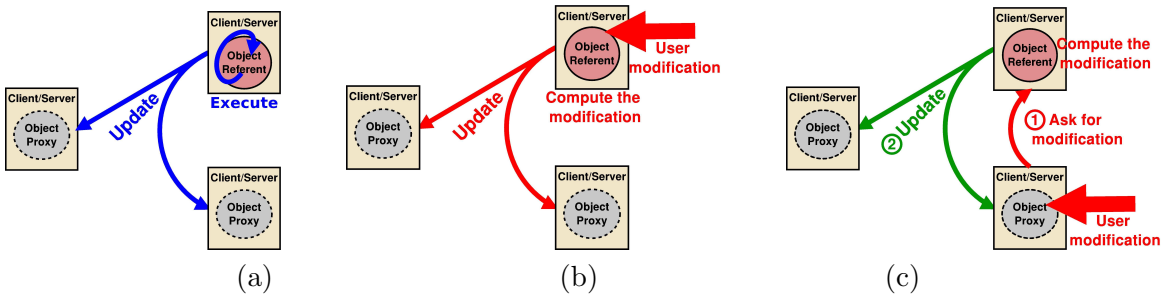


Figure 1.8: (a) Executions of object behaviors, (b) direct modification of an object through its referent, and (c) modification of an object through on of its proxies in OpenMASK.

CAVERN [LJD97] also uses a kind of referent/proxies paradigm for data distribution. Each object is defined by a "local key" on one node and "remote keys" on the other nodes. All these keys are linked together over a communication channel, so any modification of one key is automatically propagated to all the others.

Similarly, Schmalstieg et al. [SRH03] proposes to use referent/proxy paradigm to distribute a shared scene graph. Each node only replicated a part of the scene graph according to its location in the virtual world. Nongraphical application data are also embedded in the scene graph. For each object, a particular node is responsible for processing its data. A migration mechanisms can be used to change this node.

All these hybrid solutions mix features of the centralized world and features of the totally replicated world to meet particular requirements of consistency and responsiveness. The partially replicated world enables CVE systems to make a trade-off between the advantages and drawbacks of the two other data distributions. For example, replicating only the necessary objects on each node avoids replicating all the data as in the centralized world, and it makes the system as responsive as in the replicated world. However, these hybrid solutions also have some drawbacks: in the previous example, the consistency is still hard to maintain.

1.2.3 Preliminary conclusion about system architectures for CVE

A universal solution, which would meet the requirements of all CVE systems, does not yet exist. Existing CVE systems have chosen the most adapted data distribution mode to meet their requirements, and they have done a trade-off between CVE consistency and system responsiveness. So each solution has some advantages, but also some drawbacks. It would be interesting to combine

the advantages of each solution to deal with several kinds of application and several network capabilities. Contrary to the network architecture that is often imposed by the technical requirements, the data distribution can be adapted according to the application requirements (collaboration, interactivity, etc.), the tasks that users are performing during a session, and the functions that the objects fulfill in the virtual environment.

1.3 A new adaptive data distribution model for consistency maintenance in CVE

1.3.1 Introduction

We propose an adaptive data distribution model to deal with several kinds of requirements imposed by various applications and different network constraints. This model makes it possible to dynamically change the data distribution during a collaborative session to adapt it to the tasks that users perform in the virtual environment. The choice of the data distribution can be made at the object level rather than at the application level, because all the objects of a virtual environment do not necessarily have the same need for consistency.

1.3.2 A new adaptive data distribution

Some applications require a good responsiveness to user's actions, while some others require a strong consistency of the virtual environment. Moreover, during the same collaborative session, some manipulation tasks require a good responsiveness, while some collaboration tasks require a strong consistency. Finally, in a same virtual environment, some objects shared by several users can require a strong consistency, while some other objects do not. Consequently, it would be interesting to adapt the data distribution to each particular case, especially when the network capabilities are limited (low bandwidth, high latency). So we propose an adaptive data distribution model to reach the best trade-off between consistency and responsiveness according to the requirements of each shared object and according to the network constraints occurring at run-time.

This adaptive data distribution model is based on a referent/proxy paradigm that enables our CVE system to:

- implement the three modes of data distribution,
- define a particular data distribution mode for each object,
- dynamically change the data distribution mode individually for each object.

On each node, an object is represented by a referent or a proxy. A referent stores the application data of the object, executes the object behavior, and processes the modification requests for this object. According to the data distribution mode, a referent can also send update messages to its proxies on the other nodes. A proxy only stores the application data of the object and updates this data when it receives update messages from a remote referent (see Figure 1.9). So a proxy performs no processing locally, but it keeps an up-to-date state of the object and transmits the modification requests from the user of its node to a remote referent. Even if the proxy has the same inputs and outputs as a referent, it is necessarily controlled by a remote referent.

This first referent/proxy distribution (see Figure 1.9(a)) makes an easy migration of the referent possible (see section 1.3.5), because all the application data is already on the proxy. When a proxy must become a referent, it has just to start to process the object behavior and the modification requests. No additional data transfers are required, but the application data are replicated on all the nodes. If we do not want to replicate this data, it is possible to use a second referent/proxy distribution: a proxy can directly update the object outputs (visual representation, sound, etc.)

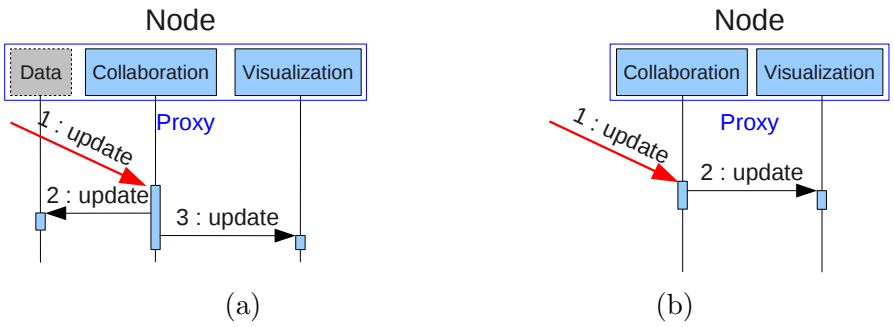


Figure 1.9: Proxy (a) stores up-to-date object data, contrary to proxy (b) which only updates the visual representation.

when it receives an update message from a referent, but it does not store any data in the application part (see Figure 1.9(b)). In this case, application data is not replicated, so it requires transferring all this data to achieve a migration of the referent.

1.3.3 The three main modes of data distribution

According to the location of the object referent, three modes of data distribution can be implemented without major system modifications. Each data distribution mode meets particular requirements of consistency and responsiveness. In this part, gap in consistency (GC) and interaction latency (IL) are quantified in an abstract way according to the value L of network latency. This network latency L corresponds to the time needed to transmit a message between two nodes or between a node and a server. To simplify the explanations, we assume that processing delays of events (updates, modification requests) are negligible in comparison to network latency. Actual measurements of interaction latency and gap in consistency can be found in [FDGA10a].

1.3.3.1 Centralized mode

To achieve a centralized data distribution for an object, only one referent is put on a server. All the other nodes have a proxy that is controlled by the referent on the server. Behavior and modification requests are processed on the server, and then update messages are sent to all the nodes. So GC is quasi-null if all the nodes have similar network connections, but IL is about $2L$ during user interactions (see Figure 1.10).

1.3.3.2 Hybrid mode

To achieve a hybrid data distribution for an object, only one referent is put on a particular node. All the other nodes have an object proxy that is controlled by this remote referent. For behavior execution and modification processing, GC is quasi-null for all nodes, except for the node which owns the referent: it perceives the object state with an advance of L in comparison with the other nodes. IL can be quasi-null if the user interacts with the referent (see Figure 1.11(a)), but it can be about $2L$ if the user interacts with a proxy (see Figure 1.11(b)). However, migration mechanism can be used to move the referent to the interacting user node (see part 1.3.5)

1.3.3.3 Replicated mode

To achieve a replicated data distribution for an object, the referent/proxy paradigm must be adapted: one referent for this object is put on each node. The object behavior is executed on each node and no update messages are sent between nodes. So the gap in consistency can be very large if

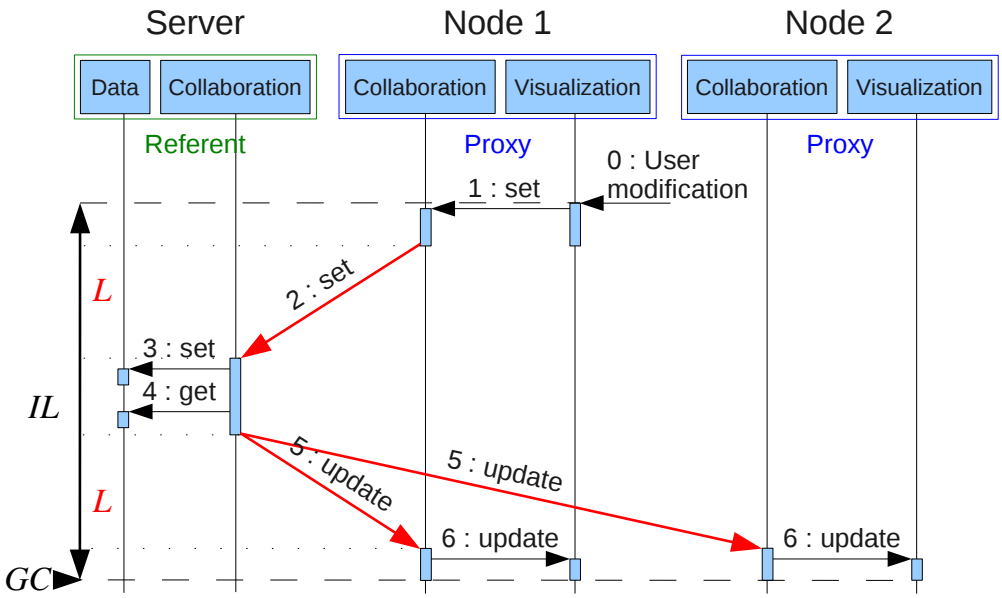


Figure 1.10: Modification of an object in centralized mode.

no synchronization mechanism is used between nodes. A synchronization mechanism enables object behavior to be executed at the same time on each node, and reduces the gap in consistency. For modification, requests are processed locally on the interacting user node, then update messages are sent to the other nodes (see Figure 1.12). So IL is quasi-null, but GC is about L for modifications.

This data distribution mode offers at least the same or better responsiveness during user interactions than the hybrid mode. However, to guarantee the same CVE consistency between all the nodes, a strong synchronization must be achieved, especially with objects whose behavior independently evolves in time. Processing all the object behaviors on each node and achieving the synchronization cost a lot of processing power on each node, so it can reduce the user interaction capability. Moreover, simultaneous interactions with a same object by remote users are quite impossible because each node locally computes its own modifications.

1.3.3.4 Quantitative comparison of the three modes

Data distribution mode	**user IL**	**others IL**	**GC**
Centralized	2L	2L	0
Hybrid *Referent on user node*	0 *(referent)*	L *(proxy)*	L *(ref. ↔ proxy)* 0 *(proxy ↔ proxy)*
Hybrid *Proxy on user node*	2L *(proxy)*	L *(referent)* 2L *(proxy)*	L *(ref. ↔ proxy)* 0 *(proxy ↔ proxy)*
Replicated	0	L	L

Table 1.1: Interaction latency and gap in consistency for user interaction according to the data distribution mode.

Table 1.1 summarizes values of interaction latency and gap in consistency of the three data distribution modes. This table takes into account the time between a user's action and the system response on his node (user IL), on the other users node (others IL), and the gap in consistency between these users (GC). We can see that only the centralized mode guarantees a perfect consistency, whereas the replicated mode makes low latency during interactions possible. Finally, the

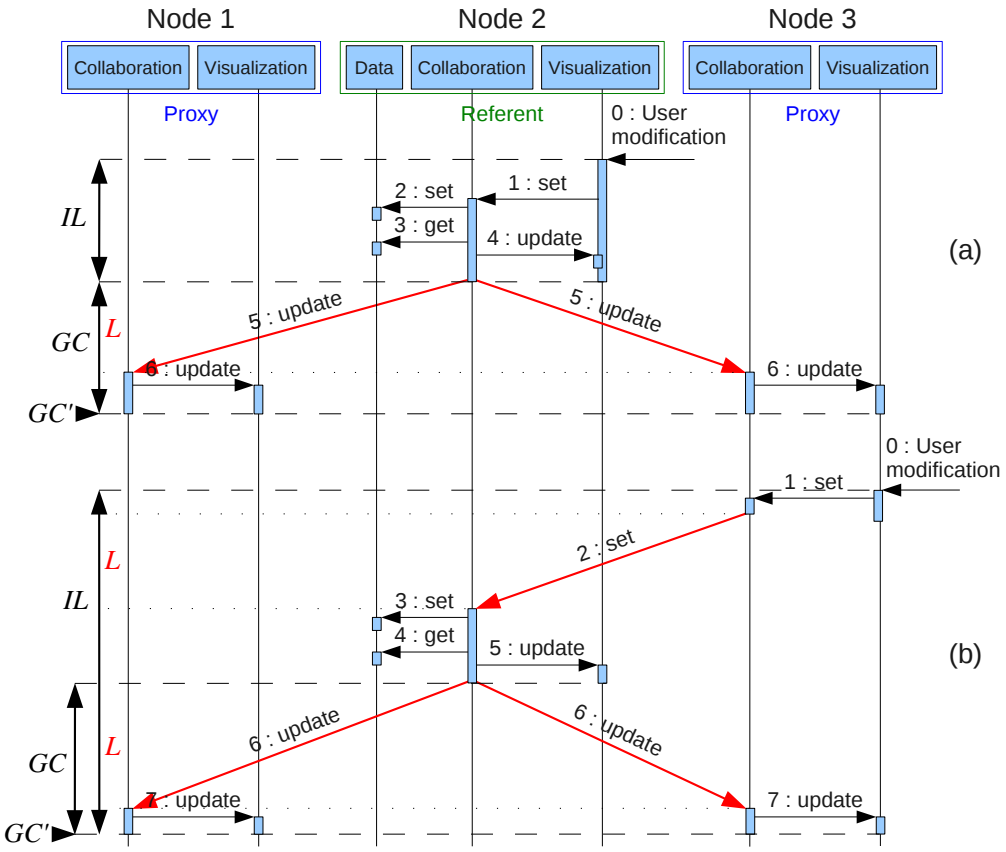


Figure 1.11: Modification of an object in hybrid mode when user is interacting with the referent (a) or with a proxy (b).

hybrid mode can ensure at the same time a good consistency and low latency interactions, but it requires to move the object referent to the interacting user node.

1.3.4 Each object can choose its data distribution mode

Using a referent/proxy paradigm enables us to choose a particular data distribution for each object, contrary to the existing CVE systems, which are designed with only one data distribution mode for all the object. In fact, for each object and on each node, a simple boolean indicates if the object version is a referent or a proxy, and consequently if this object version has to process the object behavior and modification requests or not. So the referent locations can be different for each object, and an object can also have several referents on different nodes. When several referents are used for a same object, a concurrency control must be achieved to avoid that concurrent modifications are done on each referent.

The choice of the data distribution at the object level is motivated by the fact that each object does not have the same need for consistency according to the function that this object fulfills in the virtual environment. For example an object, such as a pointer, used to show something to another user, or temporally animated objects which have to progress at the same time on each node will require a strong consistency. However, a static object decorating the virtual environment will probably not require a strong consistency. So it can be interesting that this kind of objects requires just few network transmissions in order to reduce the global communication cost. Moreover, some tools used for object manipulation will require a good responsiveness during interactions instead of a strong consistency.

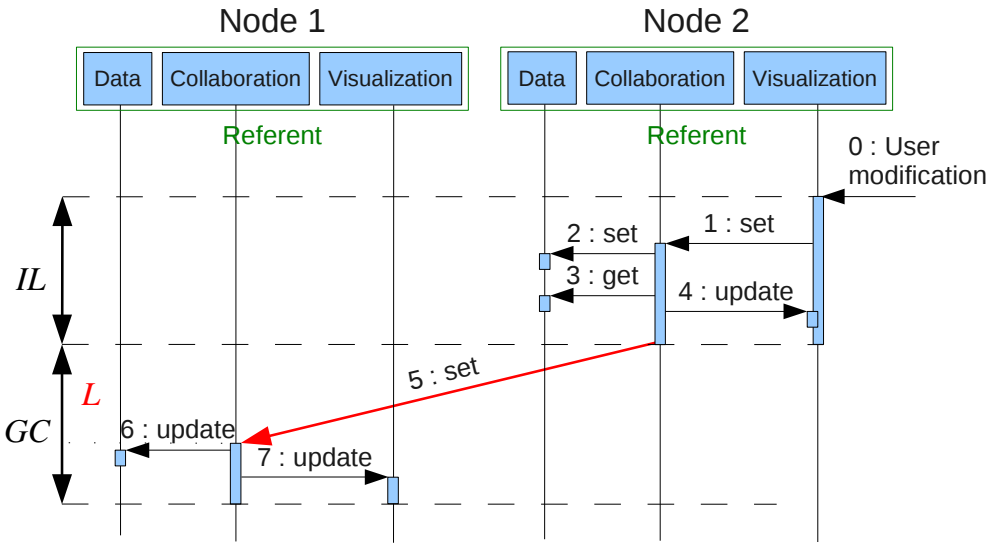


Figure 1.12: Modification of an object in replicated mode.

1.3.5 Dynamic changes of data distribution mode

With the referent/proxy paradigm, it is quite simple to dynamically change an object data distribution mode during a collaborative session. If the data is already on proxies (see Figure 1.9(a)), only the status boolean has to be changed to indicate that a proxy is now a referent, and vice versa. If the data is not on proxies (see Figure 1.9(b)), object data has to be transferred from a referent, before a proxy can become a referent. So data distribution can be easily changed from one mode to another to adapt this data distribution to user's actions or network troubles during a session. In the same way, the referent can be moved to the interacting user node when the hybrid mode is used (migration mechanism). The following examples illustrate dynamic changes for each data distribution mode:

- centralized mode → hybrid mode: if a user needs to use a 3D pointer to precisely manipulate a virtual object, the 3D pointer referent can be moved to the user node to offer good responsiveness during the manipulation.

- hybrid mode → centralized mode: if now the user wants to use this 3D pointer to show a point of interest in the virtual environment to another user, the 3D pointer referent can be moved to a server to ensure that the two users see this pointer at the same location at the same time.

- centralized or hybrid mode → replicated mode: in case of network troubles, the replicated mode can be used to reduce network communications and enables users to interact in spite of network issues.

- replicated mode → centralized or hybrid mode: if an object behavior suddenly requires high processing power for specific processing as scientific data computation, the object referent can be moved on a server or on a node with high processing power to not overload all the nodes.

Rules for data distribution changes can be determined by the object behaviors or by the application controller. So this rules could be defined in the application configuration files or in the object description files (see section 1.4).

1.3.6 Instantiation of the data distribution model for collaborative scientific visualization

This adaptive data distribution model has been used to design the Collaviz framework [DDF+10], a new CVE system dedicated to collaborative scientific data visualization. The Collaviz project aims to enable remote experts to visualize complex scientific data together by sharing a common virtual environment in an industrial context. The details can be found in [FDGA10a].

1.4 Conclusion and future work

To enable users to perform an effective collaboration in a shared virtual environment, a good consistency has to be maintained between the users' nodes. However, maintaining CVE consistency must not reduce user interaction capability by decreasing the system responsiveness. The stronger the network constraints are (low bandwidth, secured protocols), the more CVE systems must deal with the application's particular requirements to reach a good trade-off between consistency and responsiveness. So we propose an adaptive data distribution model. This model enables our CVE system to achieve three data distribution modes to deal with several applications requirements and various kinds of network connection. The data distribution mode can be individually chosen for each object according to the function that it fulfills in the virtual environment. Moreover, this data distribution can be dynamically changed during a session to adapt itself to the tasks that users need to perform in the CVE. Finally, measurements of interaction latency and gap in consistency between remote computers show that each data distribution mode has its own performance characteristics. So, adapting the data distribution according to the application requirements improves collaboration and interaction capabilities of CVE users, especially with a WAN.

Future work could focus on developing a syntax to describe rules of data distribution changes for each object. According to the user actions or to the network status, these rules could determine when the data distribution mode of an object must be changed. We could propose extensions to an XML language (such as X3D or Collada) in order to include such rules in the description of the shared virtual world.

Chapter 2

Synchronization Models for Collaborative Virtual Environments

2.1 Introduction

In addition to the system architecture and to the communication protocol, some mechanisms can be used to improve the virtual environment consistency. First, we present time synchronization mechanisms that enable each node to process CVE changes at the same time. Second, we examine different solutions used to manage the concurrent access to virtual objects.

2.2 Related work

Time is a fundamental element of a CVE. The notion of time can differ from one application to another. Delaney et al. [DWM06a] distinguish two different notions of time in CVE:

- **Absolute time:** the time of a CVE can be based on a clock periodically synchronized between each node, based on the coordinated universal time (UTC).
- **Logical or virtual time:** the time of a CVE can be based on a logical clock that is not precisely synchronized between each node as proposed by Jefferson [Jef85]. This time can be seen as an ordered event sequence. When no new event occurs, it stays the same.

The relationship between time and consistency is very important. In a perfectly consistent CVE, all the users perceive the same state at the same absolute time. However, this perfect case can never happen because of network latency. Delaney et al. [DWM06a] list different solutions to improve consistency over time according to the required responsiveness for interactions.

2.2.1 Time synchronization

2.2.1.1 Lockstep synchronization

The lockstep synchronization used in RING [Fun95] or OpenMASK [MAC+02] is the easiest way to ensure the consistency of a CVE. It consists in stopping nodes until all of them have processed the current simulation step. So, each node does not increment its logical clock until all the other nodes have acknowledged that they are ready for the next simulation step. This solution avoids roll-backs because it imposes that events are processed in the correct order. However, it guarantees consistency but not in real-time. If there are delays or losses during transmissions, the time spent to wait increases, and the system responsiveness breaks down. Furthermore, simulation steps are not necessarily constant, so the system jitter can be substantial.

2.2.1.2 Imposed global consistency

This technique consists in delaying the processing of both local and remote events. The delay is the same for all the nodes, its value is usually defined according to the maximum values of the network latency. This delay makes it possible to increase the probability that the remote events are received before processing all the events of a simulation step. This method makes possible a strong global consistency between all the nodes with an absolute clock. However, this solution introduces interaction latency which value varies according to the network characteristics. This latency is constant during the whole session.

2.2.1.3 Delayed global consistency

Contrary to the imposed global consistency, the goal of this technique is to maintain an asynchronous consistency. Users perceive the same state of the virtual environment, but not at the same time. Each event is marked with a "timestamp" (using a logical clock). Each node manages its logical clock. So the state of a CVE can be rebuilt locally with the events right order. However, this delayed consistency can disturb collaborative tasks (users may not perceive the same virtual environment at the same time).

2.2.1.4 Time warp synchronization

"Time Warp" synchronization proposed by Jefferson [Jef85] is an optimist technique, which consists in processing each event as soon as it arrives. Events are also marked with a "timestamp". When an event is received with an older "timestamp" than the event that has just been processed, the mechanism must cancel the processing of all the events with a most recent "timestamp" (roll-back). Then it processes again all these events to catch up with the current time. Moreover, it must send messages to cancel incorrect messages sent during the deprecated execution (roll-back propagation).

This synchronization method makes possible low latency interactions. However, it can only be used when roll-backs happen rarely, because they are extremely annoying for users. Several systems propose to quickly display several key-frames during the re-execution of events to facilitate the users' understanding. Finally, this method requires to store the received events to re-execute them in case of roll-back.

2.2.1.5 Predictive time management

This method proposes to predict events and to send them on the network before they occur. This concept was first proposed by Roberts et al. [RS97] in the PARADE system to manage the consistency when users collaborate through a network with inherent latency. Obviously, this mechanism can not be applied to all the virtual environment objects because some objects are not predictable. Particularly, user actions are difficult to predict. For example, PARADE uses this method for collision detection.

Events are predicted locally, marked with a "timestamp" and sent to other nodes, where these events will be processed at the appropriate time (defined by the "timestamp"). This predictive management is interesting only if the time between the sending of the predicted event and its processing is higher than the network latency. Otherwise the message will arrive too late to be processed. If the prediction is false, the system needs to make roll-backs to resolve mistakes. To minimize roll-backs in PARADE, predicted events are sent "just-in-time" by using estimations of network delays: network delays are determined by the study of the RTT (Round-Trip Time) of the network.

2.2.1.6 Server synchronization

In client/server architectures, the server can be used to synchronize events using a logical clock. In ShareX3D [JFM+08], the server maintains a "state number" for each object of the CVE. When the server receives a change for an object, it increases the "state number" of this object. Nodes also maintain a "state number" for each object corresponding to the last update messages received for this object. When a node asks the server for new changes about an object (see "long polling" in [JFM+08]), it sends the local "state number" for this object. If this "state number" is older than the server one, the server sends back an update message with the new "state number" value. Otherwise, the node is up-to-date and its request stays in standby.

This solution provides a simple way to ensure that nodes have the most recent object states, but it does not guarantee that all events will be received and processed by nodes. This is not an issue in ShareX3D because the server sends whole states of objects, so nodes can restore the state of an object with only one update message.

2.2.2 Concurrency control

The centralized data distribution ensures an implicit control of concurrent access to CVE objects, because this data can be changed only on the server. It is the same for systems that use a referent/proxies paradigm as OpenMASK [MAC+02], because only the referent of an object can be modified by users. However, when data is distributed on each node (homogeneous or partially replicated worlds), the users can access and modify objects locally before these changes are transmitted to other users. So it is necessary to explicitly manage the users' concurrent access to these objects to avoid inconsistencies in the virtual environment. When an object can be modified by only one user at the same time (non-collaborative interactions), Lee et al. [LLHL07] distinguish three kinds of mechanisms to manage the concurrent access:

- **pessimistic mode**, as in BrickNet [SSP+95]: This mode ensures that only one user can modify an object at the same time with a lock system. When a user wants to manipulate an object, he asks to become its owner. An object has only one owner. So if the object already has an owner, the user has to wait until this owner releases the ownership of this object. Only the current owner of an object can modify it. In this way, no concurrent access to an object can occur. However, when the network latency or the number of users are high, the necessary time to acquire the ownership of an object can be long and therefore introduces latency during interactions. BrickNet [SSP+95] uses the server to save which user is the owner of each object. This mode is difficult to set up in the case of a peer-to-peer architecture. Indeed, when a node requests the ownership of an object, it must ask all the other nodes if they own this object.
- **optimistic mode**: This mode enables users to modify objects without checking the potential concurrent access on these objects. So users can have low latency interactions with these objects. However, when a conflict occurs, it is necessary to make a correction. It can be complex and also requires that users perform their actions again.
- **prediction based mode**, as in PARADE [RS97] or ATLAS [LLHL07]: This mode consists in predicting for each object which users may manipulate this object to prioritize these potential owners. If the prediction is false for a user (he is not interacting with the object), it gives the ownership of the object to the next user on the list of potential owners. Generally, this prediction is based on the position and the user behavior (where they move, where they look, etc.).

In several applications, it may be useful to enable several users to manipulate a same object at the same time. Margery et al. [MAP99] classify collaborative interactions into three categories:

- Only one user can manipulate an object at the same time. So the previous modes of concurrency control can be used.
- Several users can modify simultaneously independent parameters of a same object. So the previous modes of concurrency control can be adapted to each parameter of the objects.
- Several users can modify simultaneously co-dependent parameters of a same object. Other mechanisms must be used to combine user actions to modify appropriately the object.

2.2.3 Network delays and side effects

The network affects directly the performance of distributed virtual environments systems. For example when interacting remotely, a user can take the control of a virtual object and manipulate it. Low latency offers a real time interaction by minimizing the delay between user's actions and the object's responses that may be on a very distant site. If a data transmission problem arises on the network, this problem will directly affect the remote interaction. The most frequently types of problems envisaged are the delay when transmitting data over a network and some little disconnections from time to time due to dynamic rerouting systems. If the disconnection time of a site increases during a distributed virtual simulation, the consequences can be catastrophic.

A network disconnection can drive different kinds of perturbations to a shared distributed virtual reality session. Whatever the kind of the data model the virtual reality system is using (centralized, distributed, ...) we can not avoid a periodic communication between all the sites that are sharing the same virtual world, especially when users interact with objects of the shared universes. So, from the interactivity point of view, when a network disconnection occurs, the user's interactions that exist on the disconnected site are not seen any longer by the other sites. This same user is not able any longer to see the interactions of the other users and it also produces collaboration breakdowns.

When an object is evolving linearly, we can tone down the communication problem between different sites by using a prediction system that achieves local computing to estimate for example a future position of a moving object. NPSNET implements this method using the "Dead Reckoning" algorithm [MZP+94]. But when we are facing a complex evolution or totally unpredictable actions like users' interactions with virtual objects (interaction may depend on user skills or mood), such prediction systems are unable to manage the situation in a virtual world, so we find ourselves powerless facing a technical challenge. Some other systems like SPIN [DDS+99] duplicate all universes objects and perform parallel calculations locally on each site. In this case network delay is not meaningful to a virtual session from the animation point of view, but this architecture does not resolve the main problem when objects are interactive because the possible interactions of a user can not be "duplicated", so we can not avoid the anomalies provoked by delays and disconnections. OpenMASK [MAC+02] implements a predictive architecture based on the concept of referentials and mirrors, which is similar to the distribution concept in NPSNET. The evolution of a mirror relies completely on updates received from its associated referential, so it is possible to detect the presence of a network problem (if any) referring to referentials and mirrors points of view [ZD03] because it is possible to detect that some mirrors do not receive regular updates from their referential.

2.2.4 Providing awareness of network troubles

In [VGB99] Vaghi et Al. have presented an experiment of a collaborative two players ball game where one player was subjected to an increasing amount of delay. They observed that as the network delay increases, the users (being aware) modify their strategies in an attempt to cope with the situation. Fraser et Al. have made a step forward in [FGV+00] by giving visibility to what they have called "delay induced phenomena". They have implemented a system that estimates the

maximum difference between the "objective" position of a user avatar, and where another might perceive it to be, according to network delay times between the users and speed of motion. For example, in case of network delay, an avatar is shown surrounded by a sphere that represents all uncertain possible positions, and this sphere evolves according to the network delay.

In [ZD03] we have presented two different methods to make network troubles visible and also to make a user aware of possible inconsistencies in the virtual environment. The first method is a marker system that marks all mirror objects existing on a particular site to make a user aware that these objects are neither interactive nor updated anymore. The other method is the creation of an echo object that is associated to each mirror object in the virtual world. This echo is visualized at the same place than the original object's place. In case of delay between the process of the referential and the process of its mirror, we can see a spatial gap between the motion of the referential and the motion of the echo associated with this mirror that shows in this case the position variation between these two processes. This last method presents some limitations when a large number of sites are participating to a simulation due to the very important number of echoes, which will be harmful to visualization.

2.2.5 Conclusion

We have presented mechanisms that enable CVE systems to achieve a time synchronization between users, to manage the concurrent access to the data of a CVE, and to manage network delays.

We have contributed to this field by proposing strong synchronization mechanisms in the OpenMASK framework [MAC+02] (the successor of the GASP framework [DM00b, DM00a]) and the Collaviz framework [DDF+10].

Next sections present our propositions to manage several groups of synchronization to enable all the users to interact in the CVE even if they have hardware limitations such as low processing power or low network capabilities, to make the users aware of the network troubles, and to allow object migration from one site to another.

2.3 Managing network delays with OpenMASK

As mentioned above, the default time synchronization in OpenMASK is a lockstep synchronization that ensure a strong consistency for a CVE. This mechanism has been enriched to also support a weak synchronization, which can be useful when some troubles occur on the network. This mechanism comes with the possibility to make the users aware of the troubles that are occurring on the network, and to make OpenMASK objects migrate from one site to another.

2.3.1 Detection and awareness of network troubles

Whatever the kind of the data model the virtual reality system is using (centralized, distributed, ...), we can not avoid a communication between all the sites that are sharing the same virtual world, especially when users interact with the shared universes. One of the common synchronization methods in distributed systems is the use of periodic synchronization messages. This method ensures a hard real-time synchronization between all sites. Hard real-time applications require a response to events within a predetermined amount of time in order to function properly. Network delays or troubles will deny events and updates from coming in time, which will cause the breaking of the real time concept. In a distributed virtual reality simulation context, the consequence of breaking real time may be one of the two following scenarios: freeze the whole virtual world on all sites until a synchronization message shows up, which is very harmful for the session especially if the delay is important; or let the distributed virtual world goes on even in case of delay or

disconnection. This second scenario will split up the virtual world into several parallel worlds, due to different users interactions on the same virtual object, which is a very bad choice.

In this section we present our proposition to manage these network troubles, which is a mix of the two scenarios. We choose to let the simulation continue by freezing only the parts of the world whose state is uncertain for consistency considerations. We let the virtual simulation evolve even in case of presence of non updated values due to latency or disconnection. So, the virtual world is not out of use while waiting the reception of updates as it is in hard real time synchronization based distributed systems. Even objects interactivity will be preserved but only for objects that are calculated locally. Remote calculated objects lose their interactivity as long as they are disconnected.

2.3.1.1 Detection of network delay or disconnection

Usually, all participant sites to a virtual simulation using OpenMASK are sending each other synchronization messages, used to synchronize an object with its distant mirrors and to carry the updates from a referential to its mirrors. A synchronization message may even contain only a dating element in case of no new changes in the virtual world. Once a participant site is not receiving synchronization messages from one or more sites any longer, these sites will be declared as disconnected. Each site conserves a list of disconnected sites that is not necessarily the same on each site. The synchronization message time-out threshold has to be carefully determined according to the characteristics of the network, otherwise it could downgrade the system performance, either by too frequent creation of unnecessary echoes or by detecting the troubles too late.

2.3.1.2 The awareness provider system

In order to make the participants aware of any possible network trouble, we have implemented, in the kernel of OpenMASK, some services that provide useful informations to the higher level applications. In [ZD03] we did a first step for realizing an information system provider implemented in the kernel of OpenMASK. Next we will detail the limitations of this first implementation and we will explain the solutions that we have proposed and realized to enhance our system.

The echoing system. We want to make users aware that sometimes their interactions are not seen by other users as they should be, because of network delay or disconnection. The idea is to use an echo object that represents the state of its associated real distant object. Initially, the echoing system had been implemented in a static way.

On each process, for each mirror object in the virtual world, we associate an echo object that follows the motion of the mirror. The echoes association is made in a special configuration file where we specify manually which objects in the world will have echoes. The echo is a referential, it will have mirrors on the other sites as all OpenMASK simulated objects and it is visualized at the same place than the original object (e.g. the referential of the mirrors it is echoing). Physically the echo has the shape of its original object but is a little bit smaller and half-transparent so that we can not see it when a simulation is going on normally. We can associate as many echoes as we have mirrors in the virtual world. When the delay between the process of the referential and the process of its mirror is important, we may encounter some inconsistencies between the state of the two processes. For example, in case of spatial motion we can see a gap between the motion of the referential and the motion of the echo associated with its mirror, as illustrated in Figure 2.1 (a)). In case of a disconnection, the echo associated with the mirror existing on the disconnected site is frozen on the screen and it does not evolve any longer. This means that the mirror concerned with this echo is not receiving updates any longer because of the disconnection.

The disadvantage of this method is that each referential object of a scene will have as many echoes as there are mirrors (participant sites). This will overcrowd the scene especially when moving

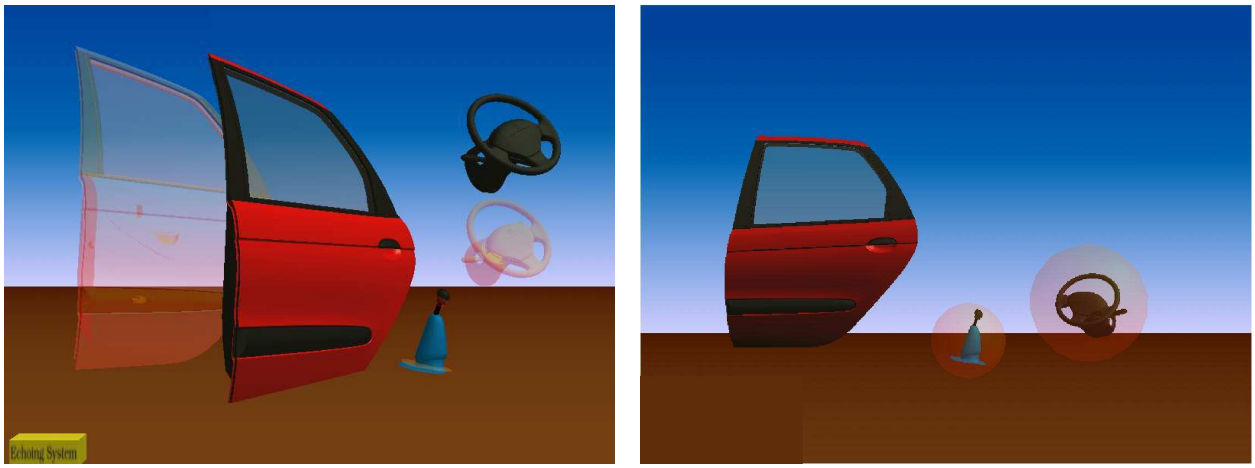


Figure 2.1: (a) The Echoing System and (b) the Marker System.

objects while interacting, and it will increase the calculus time that the system will need in case of a huge number of users. So we have decided to change the whole way of creating and managing echoes: a better solution to the above cited limitations is to create local echoes dynamically in an automatic way.

The first approach is to activate dynamically the creation of echoes after the detection of a first network perturbation, and disable the echoes as long as the distributed simulation is going on normally. This method is a clear amelioration relatively to the static creation method, but what if the first perturbation encountered during a simulation was the loss of a participating site? In this case it is too late to create echoes because we are not able to communicate with this site any longer.

We resolve this problem by providing a new concept that avoids the creation of the referentials echoes on the disconnected site: local objects. Once a site detects the lost of another distant site, it activates locally the creation of local echo objects that appear exactly with the current states (position and orientation for example) of existing referentials. Only one simulation step time separates the physical disconnection of a site from the creation of associated echoes on other sites, so the lost of the last exact value is not really significant (in the order of few milliseconds). Echoes may also appear with the current state of some mirrors in the scene. Actually the dynamic echoes creation system detects not only referentials existing on a site but also mirrors that have "brothers" located on a disconnected site. "Brother" mirrors are mirrors associated to the same referential. This way, a user becomes aware that his interactions with a mirror are not perceived any longer by some other users.

This last approach has been implemented in the kernel of OpenMASK [DZ06b] and could be extended to present more informations about disconnected or delayed sites by displaying the name of the disconnected site above the associated echo, or may be by using different echo colors in case of a few number of participating sites. Each color could then represent a particular site.

The marker system. In [ZD03] the marker system is a very limited service that marks a specific kind of objects existing on a particular site (for example the mirror objects): we use a 3D button that launches the marker system once pressed by the user. The marker system surrounds all mirror objects (or may be another kind of object) by a half-transparent sphere that makes it clear to the user that these marked objects are not holding the last updated values (Fig. 2.1 (b)). The limitation of this version of the marker system is that it was not able to be more specific by designing only mirrors damaged by a delay, so it was marking all the mirrors.

We have extended this system in order to be more specific about objects that we need to mark,

as detailed in [DZ06b]. For example, the new marker system is able now to mark only the mirrors associated to referentials that exist on a disconnected site without marking all mirror objects. This new realization offers two main advantages. First we can give to the user a very specific idea concerning the disconnected sites. Second we do not charge the scene by undesirable marks unless the user really wants to mark all the objects belonging to a particular type. We can take also a supplementary advantage from the fact of surrounding frozen mirrors, which is the protection from the user attempts for interaction. Actually, even if the user is no longer able to interact with a disconnected mirror, all his attempts are stocked in a special PVM buffer [1]. Once the site holding this mirror is reconnected, all orders stocked in this buffer are transmitted to the referential that may be confused at this time because of some contradictory orders. So, as we surround mirrors with non-interactive objects, it prevents order attempts issued from 3D direct interaction of being transmitted to the PVM buffer.

2.3.2 Migration of virtual objects

As we have mentioned earlier in the introduction, beside detecting and visualizing network troubles to the users, we want to provide them a rescue technique based on virtual objects migration. Some few virtual reality systems like AVIARY [WHH+93, SW94] and WAVES [Kaz96] have implemented object migration, but only to ensure load balancing.

In our case we use object migration to ensure a non-interrupted control of a specific object chosen by the user. Let us remind the reader that, on each site, when network troubles occur, a user can only control referential objects. Mirror objects lose their interactivity as they perform no calculation and receive their updates from distant referentials. If a user is interested in keeping the control of a specific object or a set of objects, he can claim the need of these objects to the migration system that ensures that the user will own these objects locally on his site.

We have implemented the object migration system at the kernel level of OpenMASK [DZ06a], it migrates an object by changing the state of its mirror and referential, because this ensures a good continuity of the shared virtual environment. For example, to migrate an object from $site1$ to $site2$ the migration system changes the state of the mirror of the object on $site2$ to make a referential of it, and then it changes the state of the referential on $site1$ to make a mirror of it. If no mirror exists on $site2$, the migration system will first create one. This way there is no destruction of existing objects when they migrate.

We have studied two different possibilities allowing to switch on and off between referentials and mirrors. The first solution can be achieved by implementing the migration thanks to an internal mutation of the object. The second solution enables migration by changing the nature of the object.

The first method consists in reconstructing the internal structure of referentials and mirrors so that they implement exactly the same interface. This means that a referential will contain the same functionality than a mirror, in other terms it is considered as a referential and mirror at the same time. Referential or mirror interface will be enabled or disabled accordingly to the need of the migration system. The advantage of this method is that the mutation of an object from a referential into a mirror is very easy to realize at run-time and conversely. The disadvantage, mainly because of our OpenMASK context, is the important structure modifications that may change deeply the software architecture of a simulated object. Moreover this will generate a heavy object structure that contains referential interface and mirror interface at the same time.

The second method changes neither the structure nor the interface of referentials and mirrors. Migrating an object by changing its nature consists in removing the manager of an object and replacing it without destroying the object itself. For example, to transform a mirror into a referential, we only destroy its mirror manager and we replace it by a referential manager. In case a user has given some specific behavior to an object, he can migrate this behavior by redefining two extra

[1] OpenMASK's distributed version rely on PVM [PVM]

methods *emigrate()* and *immigrate()*, which are kinds of serialization and deserialization methods that allow to determine all the important information to transmit from the old referential to the new one, within a dedicated message.

Although the first method is the most efficient one, we adopted the second method to implement object migration within OpenMASK, because the first method would have imposed too much modifications within the OpenMASK kernel. But the first method has been the model for the data distribution and migration of the Collaviz framework.

2.4 Object migration with Collaviz

As detailed in section 1.3.4, each Collaviz object can have its own distribution mode, and as explained in section 1.3.5, this distribution mode can be changed dynamically at run time, which makes object migration possible simply by switching its distribution mode from referent to proxy (or the inverse). This is the most efficient migration mechanism presented in the previous section.

Collaviz proposes also to create objects that can be local to a site, not shared with the other sites. So the marker system and the echoing system of OpenMASK could easily be implemented in Collaviz.

2.5 Group synchronization with Collaviz

A synchronization can be achieved for the three modes of data distribution proposed by Collaviz. For replicated data distribution, it ensures that the object behavior is executed at the same time on all the nodes. It is the only way to guarantee consistency, otherwise each node executes the object behavior as fast as it can and inconsistencies quickly appear. For centralized or hybrid data distribution, the consistency is already guaranteed by the data distribution model, but a synchronization can be used to improve the consistency by ensuring that the update messages are processed at the same time on all the nodes.

As explained previously, the Collaviz synchronization mechanism is based on a lockstep synchronization: nodes must wait until all of them have computed the current simulation step before computing the next simulation step. A simulation step consists in (1) processing incoming events such as update messages, (2) computing object behaviors, and (3) sending resulting events. The server is used to achieve the synchronization: it sends a message authorizing the computation of a simulation step to all the nodes, and waits acknowledgments of all the nodes before sending the next message. A desynchronization of a few simulation steps can be authorized to overcome network latency.

To avoid that a node with low processing power or low bandwidth slows down all the other nodes, we propose to perform a synchronization by groups of users (see Figure 2.2):

- one group is strongly synchronized with the server,
- N groups have weaker synchronization (all the users of a same group are synchronized together),
- one group is not synchronized.

These groups can be dynamically changed during a session. When a node slows down a synchronization group, the node is moved to a group with a weaker synchronization. Consequently, the slower nodes have a delayed state of the virtual environment, but they are limited by their network capabilities or their processing power in any case. On the contrary, when a node quickly computes simulation steps according to its synchronization group, it can be moved to a group with a stronger synchronization.

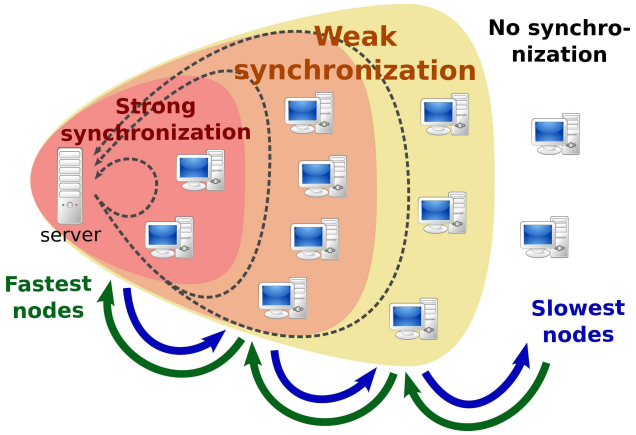


Figure 2.2: Server manages group synchronization.

Finally, the server is used to manage the concurrent access to the virtual object. It performs a pessimistic concurrency control as in BrickNet [SSP+95]: only one user can modify an object at the same time. The server stores which user has the modification rights for each object. Before modifying an object, a user must ask the modification rights for this object to the server. If another user is modifying the object, he has to wait until this user has finished.

2.6 Conclusion and future work

We have proposed some mechanisms to detect and visualize network problems while interacting within a networked virtual environment, and we have proposed to perform a synchronization by groups of users to ensure the best trade-off between synchronization and interaction latency.

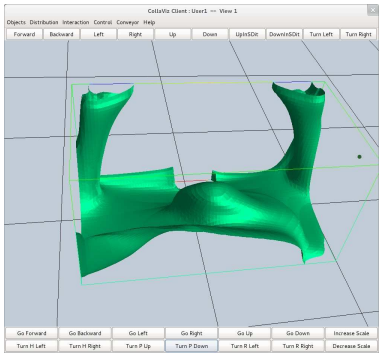
Two kinds of metaphors are used to inform users about the availability of the updates of an object: echoes for users having referentials, and marks for those having mirrors. Those metaphors are coupled with a virtual object migration mechanism implemented within the OpenMASK platform. This migration mechanism is used to ensure a non interrupted control and manipulation of an object by migrating this object onto the site of a user who wants to interact with it, since local interactions are not sensitive to a network problem.

Our contributions make possible the dynamic management of areas of interest by migrating automatically a set of objects to a particular site depending on the interest of users, for example depending on the 3D position of the user, to ensure that most of the objects he can interact with will be located on his site.

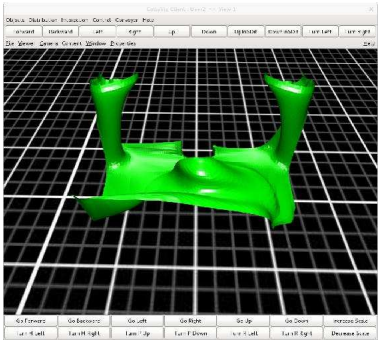
Future work could focus on making users aware of the level of the degradation of the synchronization (in which synchronization group is currently a user?), and, with the marker system, of the possible range of values that could have been taken by a parameter during a network breakdown.

Chapter 3

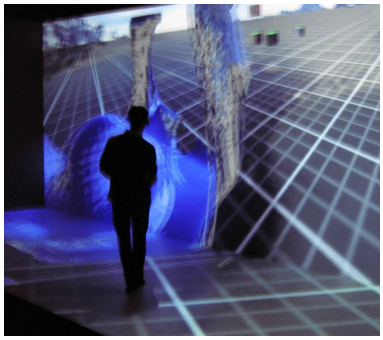
Software Architectural Models for 3D Collaborative Virtual Environments



The Java3D Visualizer



The jReality Visualizer



The Immersive jReality Visualizer

Figure 3.1: Three different visualizers sharing the same virtual environment.

3.1 Introduction

The design of 3D Collaborative Virtual Environments (CVE) merges the design of interactive 3D applications and the design of distributed collaborative applications. This task is complex because it must address 3D interaction and immersion issues as well as collaborative issues dealing with distribution, synchronization, and consistency maintenance of the shared virtual environment.

The configuration (adaptation to the hardware deployment systems) of CVE is complex because it must address various network characteristics (from high bandwidth on professional or experimental networks to low bandwidth on personal networks) as well as various displays and 3D interaction devices (from a 6-face CAVE™ [CNSD93] to simple workstations or even to simple interactive tablets). All these configurations can even be used at the same time in a single deployment in order to make asymmetric collaboration possible between remote users using different input and output devices.

To meet all these requirements, these CVE must be designed according to a software architectural model that makes it possible to adapt the distribution mode of a CVE to solve the network interoperability issues. Such a model should also make it possible to design software components that encapsulate the hardware 3D graphics requirements, in order to be able to choose at run-time the best components for each hardware configuration. Existing solutions focus either on how to

manage distribution and consistency maintenance for 3D CVE, or on how to manage independence between core functions and graphics API for 2D CSCW. For now, design models for CVE deal neither with independence to 3D graphics API nor with efficient management of distribution modes.

So we propose to merge these two main research fields in order to provide a new solution, the PAC-C3D model, which offers a better way to design and implement CVE. PAC-C3D meets two requirements: ensure synchronization and consistency maintenance of CVE, and ensure independence of CVE from 3D graphics engines to make interoperability possible between such 3D engines.

The location of the virtual environment data (i.e. geometric data, textures, etc.) is a critical decision when designing a CVE system [MZ97]. It determines which nodes (usually the users' computers) store this data, which nodes execute the processing related to each virtual object, and how the synchronization of the distributed objects is achieved. As presented in section 1.2.2, we distinguish three data distribution modes: homogeneously replicated, centralized, and partially replicated [FDGA10a], which is similar to the approaches presented in [Dew99]. A more complete overview of synchronization and data distribution within CVE can be found in [DWM06a, DWM06b, FDGA10b]. As each distribution mode has its own advantages and drawbacks, a good software architectural model for CVE should be able to manage these three main distribution modes and should be flexible enough to provide solutions for evolution toward new distribution or synchronization modes.

In this chapter, section 3.2 presents the software architectural models used in the field of HCI and CSCW. Section 3.3 presents PAC-C3D, our new software architectural model, and how its instances communicate together. Consistency maintenance is explained in section 3.4 for each of the main distribution modes. Section 3.5 describes how this model can be used to address the problem of interoperability between 3D API, for 3D graphics or physics engines. Then section 3.6 gives some implementation examples illustrating how our model faces adaptation to different distribution situations and to different 3D engines. Finally, section 3.7 presents a complementary approach for offering interoperability between several rendering engines: the Scene Graph Adapter, which is compliant with PAC-C3D.

3.2 Related work: models for HCI and CSCW

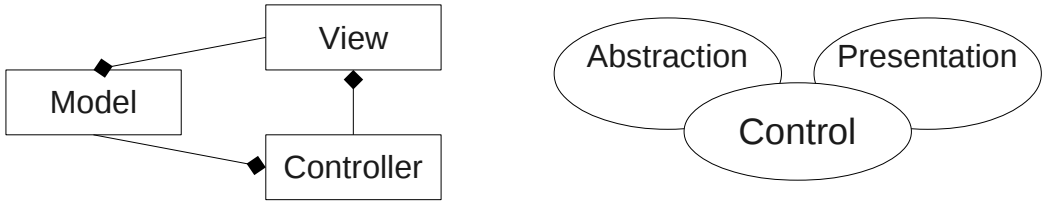
A lot of research work about architectural models for human-computer interaction (HCI) deals with separating clearly the graphics part of interactive software from its core part. Some of these models have been adapted to the context of computer-supported collaborative work.

3.2.1 Software architectural models for HCI

The most commonly used software architectural models for HCI are based either on the Model-View-Controller (MVC) model [Ree79, Gol90] or on the Presentation-Abstraction-Control (PAC) model [Cou87]. Both of them have inspired many models dedicated to particular situations: for example for Struts web-based applications (MVC-2 [Dav01]), for C++ or Java applications (Model-View-Presenter (MVP) [Pot96]) or for multi-modal applications (PAC-Amodeus [NC91]).

MVC divides interactive components into three parts: the *Model*, the *View* and the *Controller* (see figure 3.2(a)).

"The *Model* represents data and the rules that govern access to and updates of this data. ... The *View* renders the contents of a *Model*. It specifies exactly how the *Model* data should be presented. If the *Model* data changes, the *View* must update its presentation as needed. This can be achieved by using a push model, in which the *View* registers itself with the *Model* for change notifications, or a pull model, in which the *View* is responsible for calling the *Model*. ... The *Controller* translates the user's interactions with the *View* into actions that the *Model* will perform. ..." [Eck07]



(a) (b)

Figure 3.2: (a) The MVC model and (b) the PAC model.

So the *Model* and the *View* of MVC can be closely coupled, contrary to the formal separation achieved by PAC between these two kinds of components.

PAC divides interactive components into three parts: the *Presentation*, the *Abstraction* and the *Control* (see figure 3.2(b)). At first look, one could consider that PAC is just another name for MVC where *Presentation* could stand for *View*, *Abstraction* could stand for *Model* and *Control* could stand for *Controller*. In practice, the PAC components have a quite different behavior than the MVC components.

"The *Presentation* defines the concrete syntax of the application, i.e. the input and output behavior of the application as perceived by the user. The *Abstraction* corresponds to the semantics of the application, it implements the functions that the application is able to perform. ... The *Control* maintains the mapping and the consistency between the abstract entities involved in the interaction and implemented in the *Abstraction*, and their *Presentation* to the user. It embodies the boundary between semantics and syntax. It is intended to hold the context of the overall interaction between the user and the application." [Cou87]

So the *Abstraction* and the *Presentation* are not allowed to communicate directly: the *Control* acts as a mediator and filters all the communications between its *Abstraction* and *Presentation*, and with the other *Controls*.

In fact, most of the MVC-like models propose also this separation between the *Model* and the *View*, as detailed in the Oracle/Sun interpretation of MVC [Eck07], which is very similar to the PAC model.

In order to ensure a better independence between these three kinds of components, the Arch model [UIM92] proposes to add adaptor components between them (see figure 3.3). This model is also considered as a meta-model for other software models, which should follow this generic separation between facets of interactive components.

These models must now be extended to manage the collaborative aspects of CVE.

3.2.2 Models for collaborative HCI

Several adaptations of the PAC and Arch models have been proposed to cope with these collaborative features. They rely on Ellis's conceptual model of groupware [EW94] or on the clover conceptual model [LN02]. Ellis's model proposes three complementary components or models: ontological, coordination, and user-interface model. The clover conceptual model proposes to divide the services of collaborative software into three main parts: production, communication and coordination (see Figure 3.4(a)). Ontological model and the production space refer to the shared virtual objects of a CVE. Coordination model and coordination space cover the consistency maintenance in the CVE. User-interface model refers to the representation of human-computer interaction while communication space refers only to the communication between the users of a CVE.

PAC* [CCN97] (see Figure 3.4(b)) dispatches these three kinds of functions across the three PAC facets. To our opinion, this is a problem for designing *Abstractions* independently from the

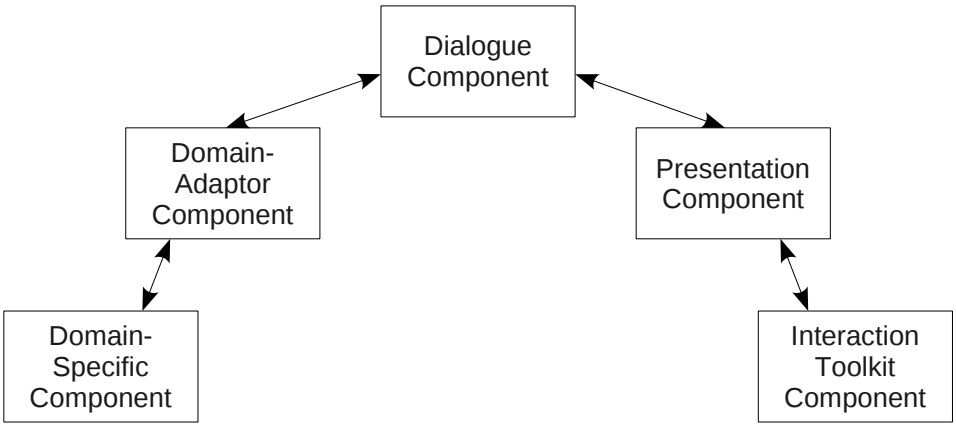
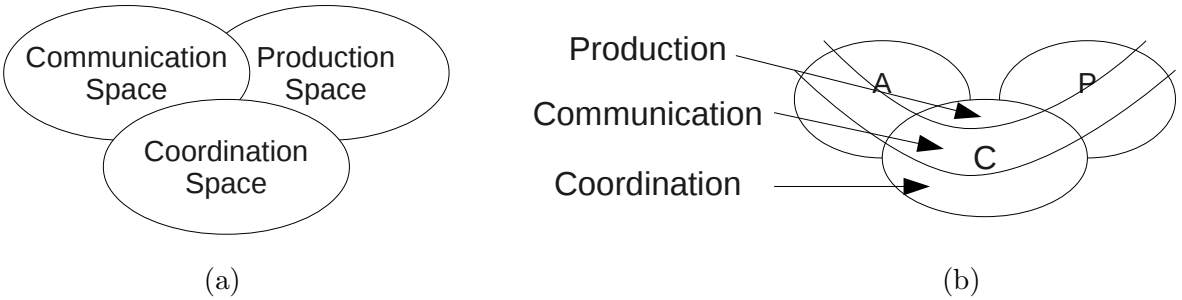


Figure 3.3: The Arch model.

collaborative aspects.



(a) (b)

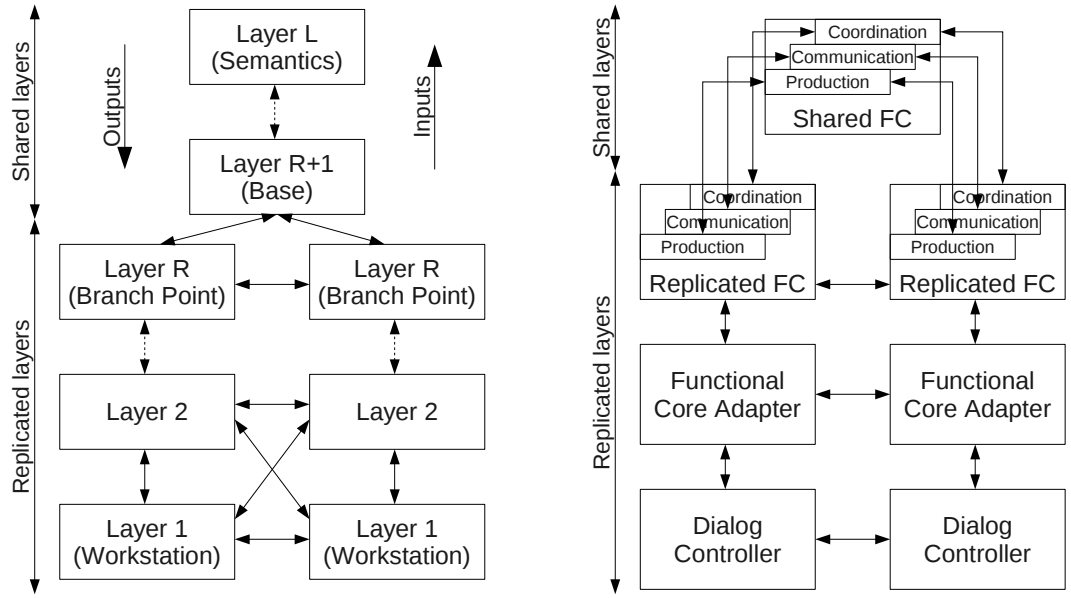
Figure 3.4: (a) The clover concepts and (b) the PAC* model.

Clover [LN02] (see Figure 3.5(b)) is an extension of PAC* that relies on Dewan's "generic multi-user architecture"[Dew99] (see Figure 3.5(a)), which is a collaborative extension of the Arch model. Here again, each unit can contain three sub-components about production, communication and coordination, especially the higher-level units that correspond to the core of the CVE.

3.2.3 Synthesis about HCI models and collaboration

Software architectural models for HCI propose to divide interactive components in three kinds of components that should be as independent as possible from each other. Some of these models have been extended to address the design of CVE, according to the clover conceptual model, but they do not address how to cover the three main distribution modes of the CVE. Furthermore, they spread the collaborative aspects over all the components of the models, which is a problem for designing *Abstractions* that should not be aware of the collaborative aspects.

This is why we need a new model for designing 3D CVE, which would ensure the best possible separation between core functions, visualization (3D graphics API and libraries) and collaboration aspects, and which would provide explicit solutions to achieve these different synchronization modes.



(a) (b)

Figure 3.5: (a) Dewan's model and (b) the Clover model.

3.3 PAC for collaborative 3D applications

3.3.1 Interfaces for independence between components

In order to make the PAC facets independent from each other, we choose a special interpretation of the PAC model that proposes interfaces to specify the features of each facet of the model. An important feature of this model is that the *Control* is a *Proxy* (GoF207)[Gam95] of its associated *Abstraction*.

In Figure 3.6 we present a new interpretation of this model in order to allow the presence of several *Presentations* associated to the same *Control*. Each virtual shared object will be described through 3 interfaces:

- *Interface for the Abstraction* (**IA**): it declares the methods in charge of the object behavior and the methods allowing to set and get its attributes.
- *Interface for the Presentation* (**IP**): it declares the methods allowing to set and get the attributes of the representation of the object (for example the position of its 3D visualization).
- *Interface for the Control* (**IC**): it declares all the methods of the *Interface for the Abstraction*, as the *Control* will be used to manage the access to the *Abstraction* (the *Control* will be the proxy of the *Abstraction*) to maintain consistency between the *Abstraction* and all the *Presentations*, and some methods dedicated to the communication with its *Presentations* and the other *Controls*.

As these interfaces will be implemented by the real facets of the PAC components, at run time these facets will be instances of:

- *Abstraction* (**A**): it implements the object model and behavior, and the setters and getters.
- *Presentation* (**P**): it implements the object representation: for example it can use a 3D graphic API to visualize the object and its properties.

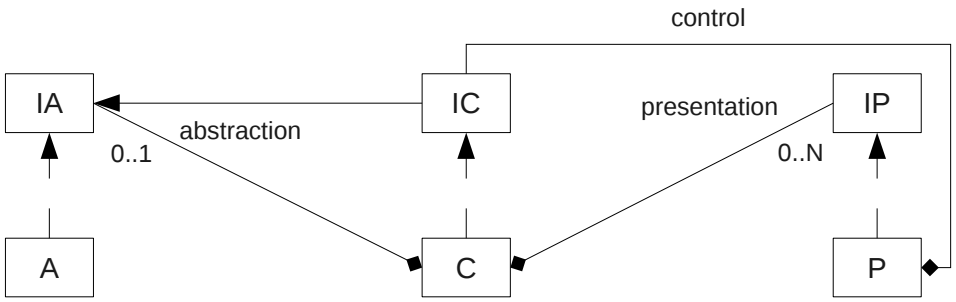


Figure 3.6: The PAC model with interfaces between facets.

- *Control* (**C**): it implements the consistency maintenance between the *Abstraction* and the *Presentations*, and it regulates the access to the *Abstraction*.

Very often, a *Presentation* is closely coupled to a 3D graphics API, but thanks to the *Interface for the Presentation*, the *Control* will be totally independent of this 3D API.

In the same way, thanks to the *Interface for the Control*, the *Presentations* and *Abstraction* will be totally independent from the implementation of the *Control*.

3.3.2 Adapting PAC to collaboration

As for the PAC* model [CCN97] and the Clover model [LN02], here again the PAC model will be the basis of our proposition, but unlike these two models, the collaborative parts will not be spread out into all the components of the model, but only into the *Control* of the PAC components.

Indeed, we consider that the objects of the production space should remain in the core parts of a 3D CVE, and that their coordination should be achieved by the *Control* of the PAC components. Last, we consider that communications between users should be either totally integrated within a 3D CVE through shared virtual objects, or totally independent of the 3D CVE, so these communication aspects are not central to a model dedicated to the design of 3D CVE. In our opinion, the distributed aspects should not impact the *Presentations* and *Abstraction* of a PAC component, in the same way that the 3D graphics details should be restrained to the *Presentations* and that the core concepts should remain in the *Abstraction*. This independence is possible thanks to the three interfaces of our model.

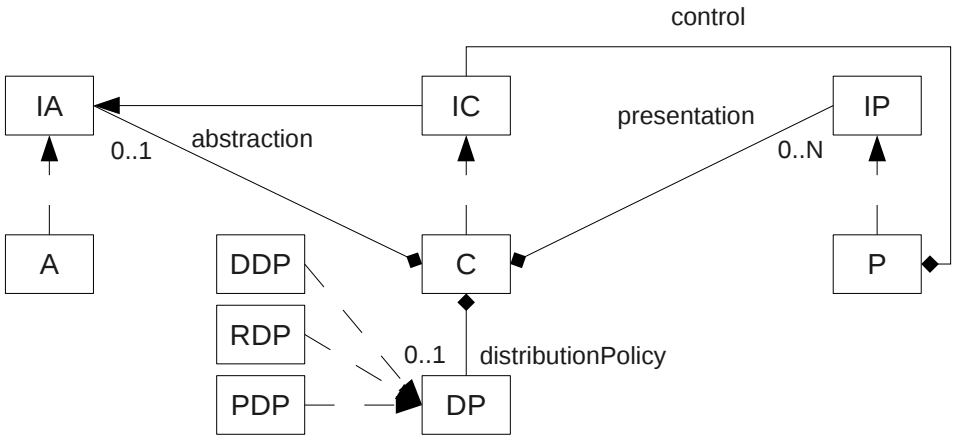


Figure 3.7: Adaptation of the PAC model for 3D CVE.

The *Control* is associated to one distribution policy, dedicated to synchronization and consistency maintenance. There are three distribution policies:

- *Referent distribution policy* (**RDP**): implementation of what is needed to manage the set messages and to distribute updates to other *Controls*: each time the value of a parameter of the object is set, this policy makes the referent *Control* send (for example using multicast) an update message to the distributed proxy *Controls* so that they can also update their *Presentations*.

- *Proxy distribution policy* (**PDP**): implementation of what is needed to transmit the set messages toward the referent *Control*, and to manage the update messages returned by the referent *Control*: each time the value of a parameter of the object is set, this policy makes the proxy *Control* send a set message to its referent *Control*, and this value will be effectively set only when the proxy *Control* will receive an update message from its referent *Control*.

- *Duplicated distribution policy* (**DDP**): same management of the update messages as the PDP, same management of the set messages coming from other objects as the RDP, and local management of the set messages due to the autonomous evolution of the abstraction.

These components will be distributed across the network according to the number of nodes in a collaborative session and to the architecture chosen for the distribution.

3.4 Dealing with distribution modes

In this section we will detail the behavior of PAC-C3D *Controls* according to their associated distribution policy, in order to show that this behavior is able to deal with the three main distribution modes for CVE.

We will consider a modification of the value of a parameter of a virtual object occurring from a presentation component where a user will have made an action upon a shared virtual object, and we will trace the subsequent communications between the PAC-C3D components and facets.

3.4.1 PAC-C3D and duplicated architecture

In a CVE with a typical duplicated architecture, for each shared virtual object there will be as many instances of *Abstractions*, *Presentations* and *Controls* with a duplicated distribution policy (**DDP**) as there are visualization nodes embedding one or several representations of the shared virtual universe.

Figure 3.8 presents a typical duplicated architecture with three nodes. The exchanges between the facets of the PAC-C3D components will be as follow:

1 An action occurs upon the *Presentation* of a virtual object, on one node, to set the value of one attribute of this virtual object. This *Presentation* does not set the attribute, but instead asks its *Control* for a modification.

2-5 This *Control* receives the set request and transmits it to its *Abstraction*. This *Abstraction* processes the set requests in its own way: the final value of the attribute of the *Abstraction* can be different from the value proposed by the *Control*, for example because a proposed value could put the *Abstraction* into an incorrect status. This is why the *Control* then asks its *Abstraction* for the effective value of the attribute and updates its *Presentation* with this new value. Then the *Control* transmits this value to the other duplicated *Controls* by sending them an update message (this sending can be synchronous but it is more efficient to make it asynchronous to allow all the duplicated *Controls* to process the update at the same time).

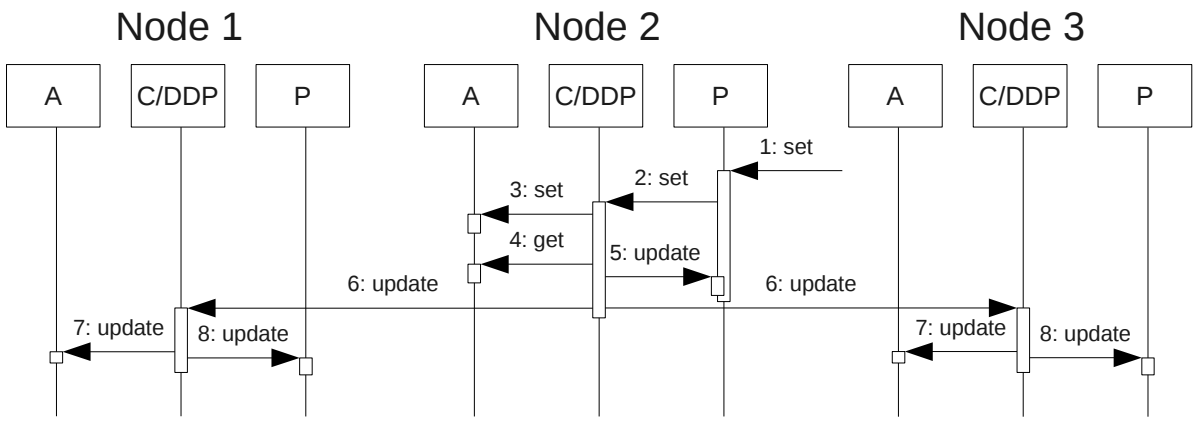


Figure 3.8: PAC-C3D and duplicated architecture.

6-8 Finally on each other node a duplicated *Control* receives the update message and asks its *Abstraction* and then its *Presentation* for an update.

As we have seen in section 1.2.2.2, the main advantage of this architecture is that when a user interacts with an object, he obtains an immediate feedback. Then all the other users may perceive the result of the interaction at the same time, with a delay corresponding to the network latency. The main drawback of this architecture is that it must ensure a strong synchronization between the nodes because of the potential autonomous behavior of some shared virtual objects. It is also quite impossible to allow several users to interact directly at the same time on a same shared virtual object.

3.4.2 PAC-C3D and centralized architecture

In a CVE with a typical centralized architecture, for each shared virtual object:

- there is only one instance of *Abstraction*, on the server,
- there are as many instances of *Presentations* as there are visualization nodes embedding one or several representations of the shared virtual universe,
- there is only one instance of *Control* with a referent distribution policy (**RDP**), without *Presentation*,
- there are as many instances of *Control* with a proxy distribution policy (**PDP**) as there are *Presentation* instances.

Figure 3.9 presents a typical centralized architecture with one server and two clients. The exchanges between the facets of the PAC-C3D components will be as follow:

1-2 On one client an action occurs upon the *Presentation* of a virtual object, and this *Presentation* asks its proxy *Control* for a modification. This proxy *Control* transmits the set request to its referent *Control* (this transmission can be synchronous or asynchronous).

3-5 The referent *Control* transmits the value to its *Abstraction*. Here again this *Abstraction* processes the set requests in its own way and then the *Control* asks its *Abstraction* for the effective value of the attribute. Then it transmits to its proxy *Control* by sending them an update message (here again this sending should be asynchronous).

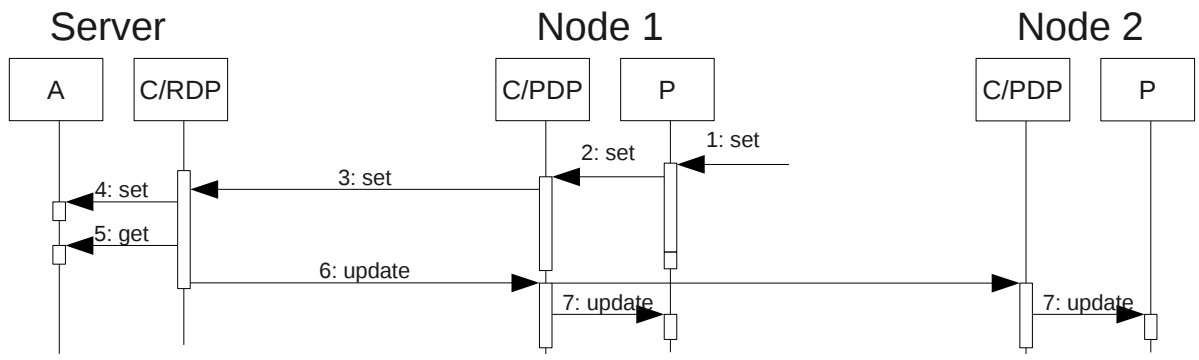


Figure 3.9: PAC-C3D and centralized architecture.

6-7 Finally on each client a proxy *Control* receives the update message and asks its *Presentation* for an update.

As we have seen in section 1.2.2.1, the main advantage of this architecture is that all the users may perceive the result of the interaction at the same time, but the drawback is that the delay for the semantic feedback is about twice the network latency. Another interesting property of this distribution policy is that it is not absolutely necessary to have a strong synchronization between all the nodes as all the behaviors are executed on the server node. Last, it is easy to allow several users to interact at the same time with the same object as the referent *Control* will be in charge of centralizing all the interactions coming from all the nodes: it can integrate all the concurrent propositions to compute a single result.

3.4.3 PAC-C3D and hybrid architecture

In order to answer more quickly to a user's interaction with a virtual object, it can be interesting to locate the *Abstraction* of this object on this user's node, which means to allow the referent to be on a client node rather than to stay on a centralized server. So we can consider that the hybrid architecture is a simple evolution of the centralized architecture where all the referents are not necessarily on the same node and where a centralized server is not absolutely necessary any longer.

In such a case, there will be two different situations while interacting with a virtual object, as described in Figure 3.10: either the *Abstraction* of the object is on the same node than the user, either it is on another node.

If the *Abstraction* of the object is on the node of the interacting user, this user will obtain an immediate interaction feedback, while the other users will perceive the interaction with a small lag mainly due to the network latency. This is very similar to the behavior of the duplicated architecture.

If the *Abstraction* of the object is not on the node of the interacting user, the user on the node of the *Abstraction* will be the first to perceive the result of the interaction, and all the other users (even the one who is interacting) will see the result of the interaction at the same time, with a delayed feedback about twice the network latency. This is quite similar to the behavior of the centralized architecture.

In both cases this hybrid solution offers the same possibility as the centralized architecture to enable several users to interact at the same time with a same object. And as we have seen in section 1.2.2.3, it also has the same main drawback as the duplicated architecture about the necessity to synchronize all the nodes because each node can be in charge of the behavior of some virtual objects.

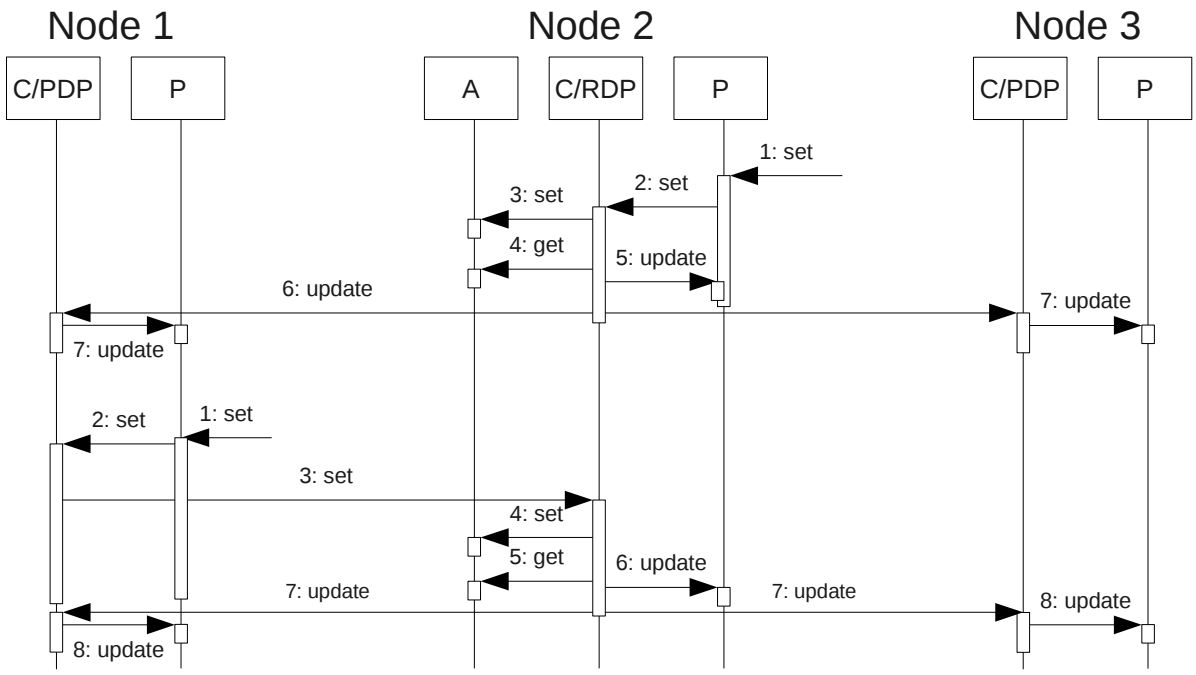


Figure 3.10: PAC-C3D and hybrid architecture.

3.4.4 Adapting distribution policies

To change the distribution mode of a system designed according to our model, for example to transform a centralized architecture to a hybrid architecture or to a duplicated architecture, we only have to change the distribution policy of the *Controls*: it impacts neither the *Abstractions* nor the *Presentations*. It is even possible to change dynamically the distribution policy of a *Control*, by replacing its current distribution policy by a new one, which allows to meet the requirements of the Collaviz system [FDGA10a].

In the same way, with a hybrid system it is possible to enable some *Abstractions* to migrate from one node to another and to change the distribution policies of the associated *Controls* to offer a better interaction to a user by placing the *Abstraction* of a virtual object on the user's node. And in the case of concurrent interaction of two users with the same object, it is better for equity to make the *Abstraction* of the co-manipulated object migrate to a third node.

Last, thanks to a precise description of the basic network services, all the network communication details are also limited to basic network policy components. These components implement the communications between the PAC-C3D *Controls* using network facilities such as RPC, RMI, TCP communications (Unicast or Multicast) or HTTP communications. So, to change the basic network layer used by the *Controls*, we only have to provide a new set of distribution policies that rely on the new network layer.

3.4.5 Creation of the shared virtual objects

To ensure an easy evolution of a CVE, we propose to use the *Abstract Factory* design pattern (GoF87) [Gam95] for object creation. This design pattern makes it possible to let the *Abstractions* create new objects without any knowledge of collaboration by asking an abstract factory to create these objects. The real instance of this abstract factory, called PAC-C3D factory, will deliver *Controls* (which are *Proxies* of their *Abstraction*) instead of *Abstractions*.

In the same way, the *Controls* must use several factories in order to create their associated

Abstractions and *Presentations*. The PAC-C3D factory will give each *Control* one factory for *Abstraction* creation, and a list (which can be empty) of factories for *Presentations* creation, corresponding to each kind of existing *Presentation* on its node. If allowed by its distribution policy, then the *Control* will ask the *Abstraction* factory to create a real *Abstraction*. Next, the *Control* will ask each *Presentation* factory to create a *Presentation*. This is illustrated in Figure 3.11 for the creation of a PAC-C3D object on one node that is hosting two kinds of *Presentations*.

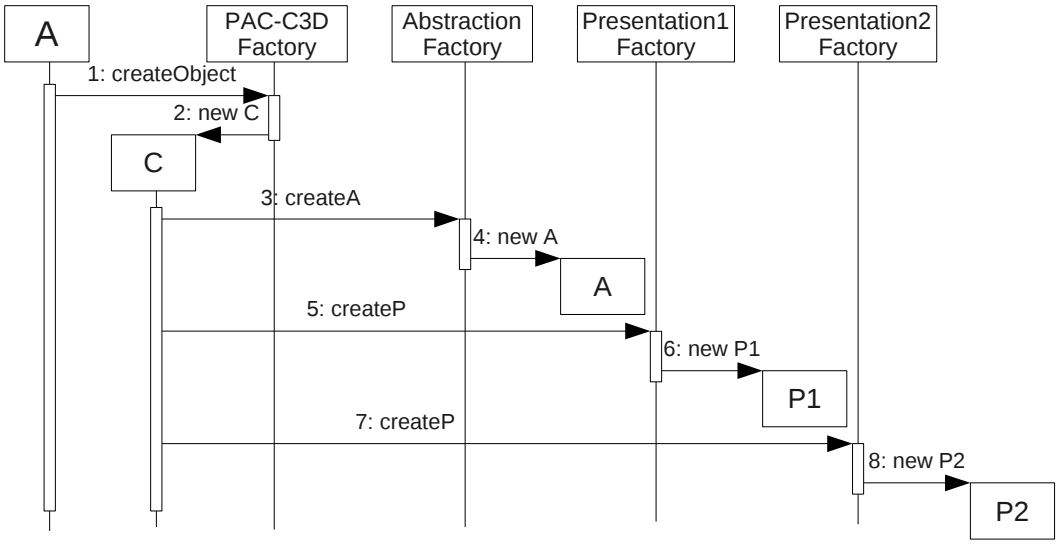


Figure 3.11: Creation of the PAC-C3D facets.

Then, according to its distribution policy, this *Control* can send a message to the other nodes of the CVE to create also local *Controls*: each local PAC-C3D factory will allow the creation of a local *Control* with the appropriate set of local *Presentations*.

3.5 Adaptation to different representations

As our model offers great independence between the facets of each PAC-C3D object, it makes it easy to provide several kinds of *Presentations* for a same virtual object.

First, for each distribution mode, the *Controls* can be associated to different kinds of *Presentations*, dedicated to a particular visualization of the shared virtual environment. For example, with *Controls* written in Java, on one node the *Presentation* could rely on Java3D [Jav] while on another node it could rely on JMonkey [JMo] or jReality [jRe]. As the implementation details of the 3D graphics API are encapsulated within the *Presentations*, for example it is also possible to use any C++ 3D graphics API without perturbing *Abstractions* and *Controls*.

To avoid code duplication, all the code relative to high-level interaction with virtual objects should be removed from the *Presentations* and written once in some *Abstractions* of PAC 3D interaction tools. But for an optimal efficiency, it is also possible to use built-in interaction and navigation metaphors that come with a 3D viewer.

Several kinds of *Presentations* can also be associated with the same *Control* in order to provide several representations of a shared virtual environment to a user. Some of these representations can also be a 2D visualization of the CVE. This can be extended to any kind of presentation, which could be non-visual, as a sound or a physical representation.

To benefit fully from "active" *Presentations* such as physics engines (for example they can react to an update because of collision detection when trying to move a 3D object), the behavior of the PAC-C3D *Controls* should be slightly adapted, otherwise the "naive" use of such engines

could introduce more latency and some small inconsistencies on other *Presentations* in the worst situation about distribution (when the physical *Presentation* is not on the same node than its associated *Abstraction*)(see Figure 3.12)). This adaptation could consist in updating first the "active" *Presentations* and taking their results into account before updating the "passive" *Presentations*.

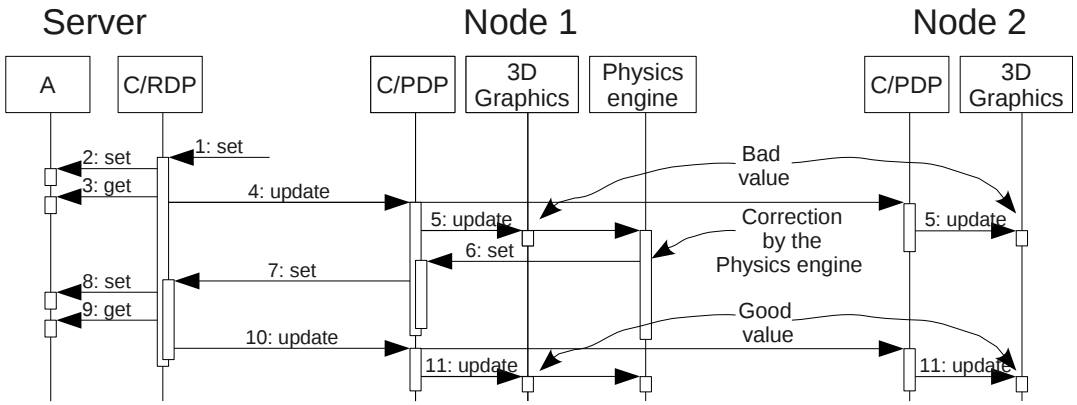


Figure 3.12: "Naive" use of a physics engine.

3.6 PAC-C3D implementation examples

In this section we will make a short point about the current implementations of PAC-C3D, then we will illustrate this model through 3 examples, in the context of the Collaviz framework [DDF+10, Col]. The first one explains in details how PAC-C3D can help the designer of a new 3D collaborative visualizer to reuse existing interaction tools. The second one describes more generally how PAC-C3D has been used to design the IIVC concept. The third one describes how PAC-C3D allowed us to integrate a physics engine within the Collaviz framework.

3.6.1 Current implementations of PAC-C3D

The first implementation of our model is dedicated to student VR projects and has already been used for two years. This simple implementation has been made with Java3D [Jav] and JMonkey [JMo] as 3D rendering engines, and implements only the referent and proxy distribution policies, with migration capabilities. The proxy distribution policy uses Java RMI for communication with its referent, and the referent distribution policy uses Multicast facilities to communicate with its proxies *Controls*.

The second implementation is dedicated to industrial collaborative scientific visualization, it has been implemented in the context of a collaborative project called Collaviz [DDF+10, Col]. It relies on Java3D (for desktop visualization), jReality (for desktop and immersive visualization) and jMonkey [JMo] (for desktop and mobile device visualization) as illustrated in Figure 3.1. The *Controls* can use the three distribution policies, and these distribution policies use either TCP or HTTP for communication [FDGA10a]. All these distribution policies can be changed at run-time. This implementation has also been coupled to the JBullet [JBu] Physics Engine which appears as another *Presentation* associated to some *Controls* of the system.

3.6.2 The 2DPointer/3DRay

Here is a full example of the benefits to use our model that shows the complementarity of the PAC-C3D separation between abstraction and presentation and of the PAC-C3D collaboration through

the controls. This exemple is the implementation of the 2DPointer/3DRay metaphor [DF09]: a 3D ray for 3D selection and interaction which orientation is computed so that the user always sees this 3D ray as a 2D pointer on the screen, to be used as easily as a classical 2D pointer, but to be seen as a 3D ray by the other users of the shared virtual environment.

We first implemented this 2DPointer/3DRay metaphor in our Java3D visualizer, this cursor was driven with mouse events provided by Java3D. We took care to clearly separate the behavior of the 2DPointer/3DRay (the computation of its orientation according to its position, that has been isolated in an abstraction component) from the Java3D presentation code in charge of the Java3D mouse events and of the 3D picking for object selection. As a first result, this 2DPointer/3DRay can be driven by any other input device able to provide a position, for example a wiimote or an ART tracking device (see Figure 3.13)).

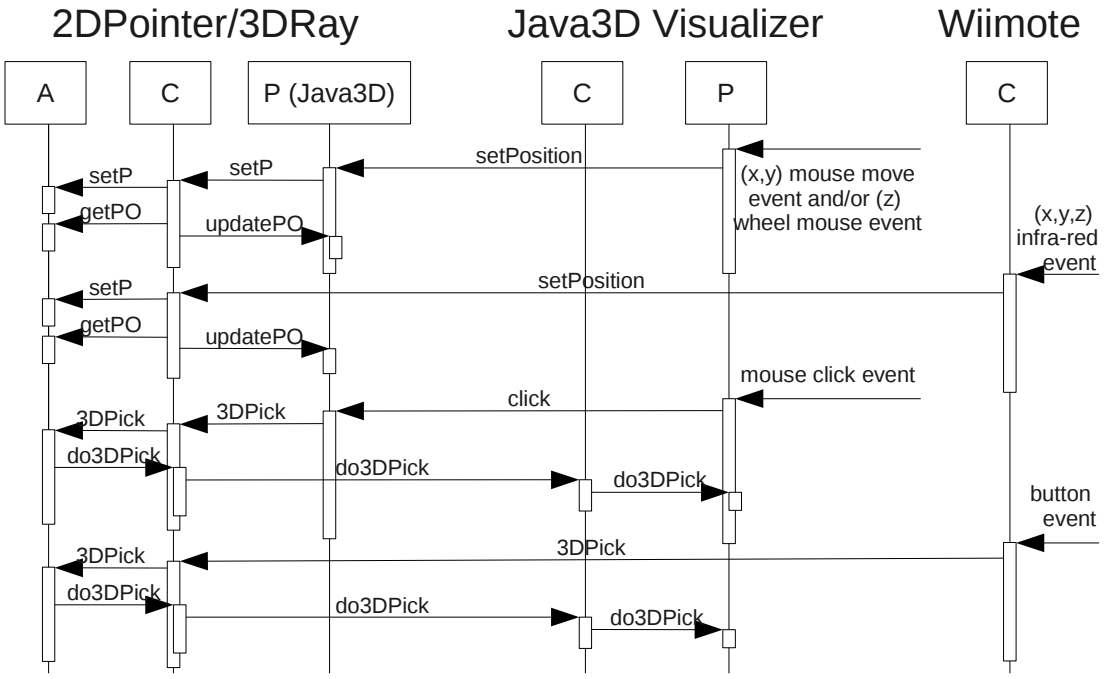


Figure 3.13: Driving the same abstraction with different input devices.

Then, we worked with jReality to provide another 3D viewer, mainly dedicated to immersive visualization devices such as workbenchs or CAVE™. As this viewer was not first dedicated to desktops, we did not want to waste time to write a small presentation component able to deal with mouse events, which would be useless for immersive situations as we would use an ART tracking system for interaction. But for testing this new jReality visualization component, some work had to be done in desktop mode, and some interaction could be useful. We decided to use the 2DPointer/3DRay metaphor driven by a wiimote (see Figure 3.14).

Once the 2DPointer/3DRay metaphor was driven by a wiimote within our jReality visualizer, the only remaining problem was to allow this metaphor to select and manipulate 3D objects. The more natural solution was to enable a 3D picking service in jReality.

If such a picking service could not have been realized with jReality, we could also have enabled the 3D picking thanks to our Java3D visualizer. As PAC-C3D has been used to design the Collaviz framework architecture, the Collaviz system allows to share a virtual environment between several visualizers, so it is possible to instantiate a Java3D visualizer and a jReality visualizer to share a common virtual environment. The abstraction of the 2DPointer/3DRay interaction metaphor of the jReality Visualizer can be instantiated on the process of the Java3D visualizer, that owns also a control component and a Java3D presentation component for this metaphor, while the process

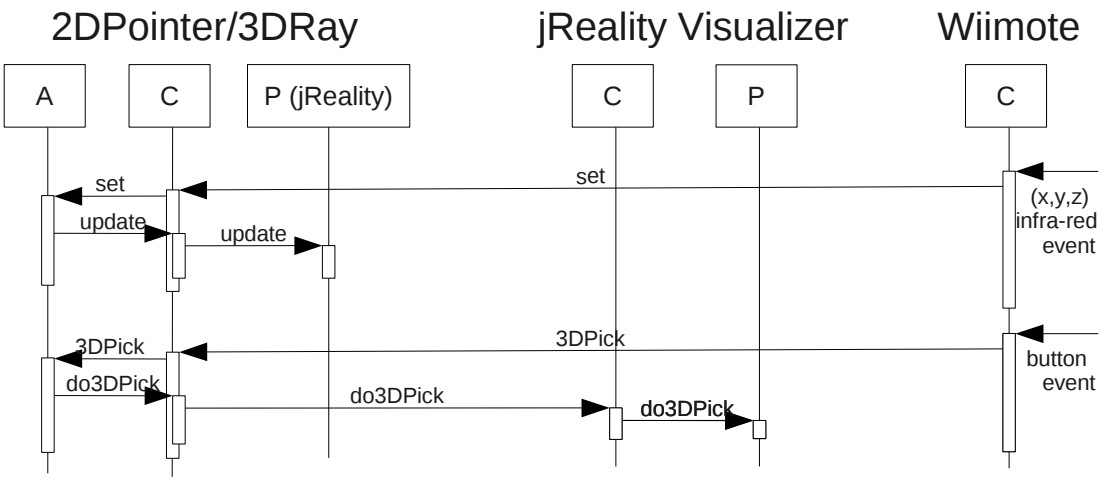


Figure 3.14: Visualizing an abstraction with another 3D API.

of the jReality visualizer owns only a proxy control and a jReality presentation component for this metaphor. As illustrated in Figure 3.15, this distribution mode would allow the proxy control on the jReality side to send the picking request to the referent control on the Java3D side, which would be able to achieve the picking thanks to the 3D picking service of the Java3D visualizer.

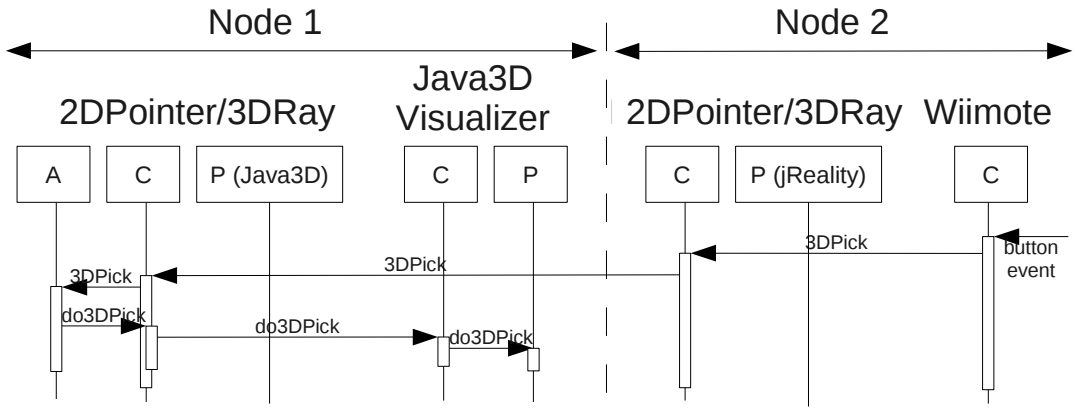


Figure 3.15: Delegating behavior to abstraction and other presentation.

3.6.3 Other interaction and navigation tools

The whole architecture of our Collaviz 3D visualizers is designed thanks to the PAC-C3D model, this enables us to provide a common architecture for navigating and interacting within 3D virtual environments that we call the Immersive Interactive Virtual Cabin (IIVC) [FCD+11] that will be detailed chapter 6. All the operators that have been proposed for this IIVC (such as dedicated navigation modes or interaction tools) are implemented within abstraction components linked to dedicated presentation in charge of the visualization of their actions upon the virtual environment through control components.

It makes it possible to use the same input devices (for example a 2D GUI or a joystick) for navigation whatever the 3D graphics API is used for a 3D visualizer: the navigation orders are sent to the abstraction of what we call a “conveyor”, which supports several virtual objects including a

virtual viewpoint, which position and orientation are changed whenever the conveyor moves in the world. These changes occur in the abstraction of the virtual viewpoint, then its control component is in charge of propagating these changes to its associated local presentation component and to its distributed controls if any.

3.6.4 Coupling a physics engine to a virtual environment

To make collision detection possible within our Collaviz 3D visualizers, we chose to integrate the JBullet Physics Engine [JBu] into the Collaviz framework. The most straightforward way to achieve this is to place the JBullet engine component on the central collaboration server process, in order to be able to provide the physics services (for example collision detection or mechanical constraints) in the same way to any Visualizer client. Otherwise, this JBullet component could be placed on any node of the shared virtual environment. So, for any virtual object for which we want to offer physics, we declare it as a physical object, with an additional JBullet presentation component, linked to the JBullet engine. Each move of the virtual object in the virtual environment will make its physical JBullet presentation move in the JBullet world (see Figure 3.16), which will allow to take into account the results (collision or constraints) provided by the JBullet engine, as already presented in Figure 3.12.

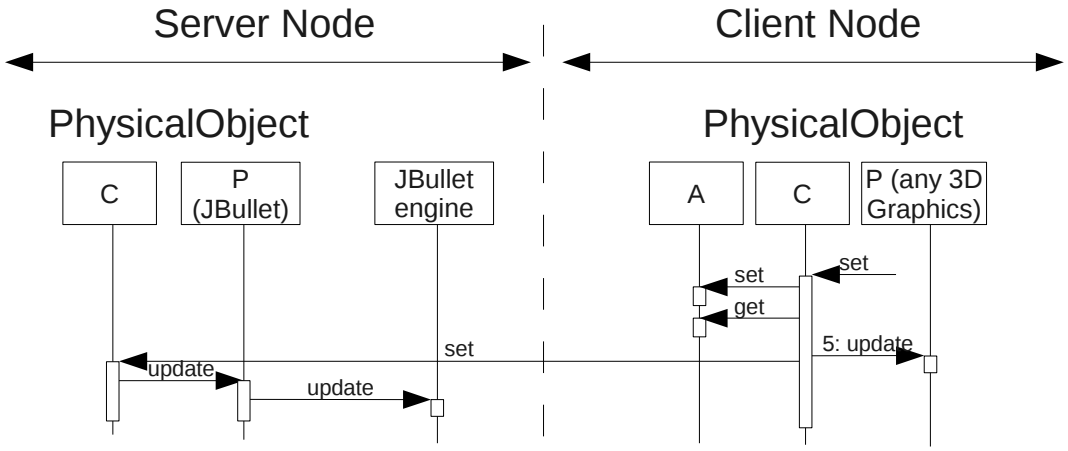


Figure 3.16: Maintaining consistency between Virtual and Physical worlds.

Once again, this allows to offer physics for the objects of the virtual universe whatever the 3D graphics API used for the 3D Visualizer: we have implemented generic physical 3D cursors that behave exactly in the same way in our two main visualizers, based on Java3D and jReality.

3.7 Another approach: the Scene Graph Adapter

In this section we present another architecture for building 3D applications that use several 3D formats for describing a virtual universe that can be rendered through several rendering engines. It is based on the Scene Graph Adapter, which aims at interfacing communication between 3D application inputs (e.g. 3D files) and output (e.g. the interactive visualization window) through the use of several API. These API ensure a strong separation between the rendering engines, they can be compared to the interface components proposed by the PAC-C3D model.

3.7.1 Concepts and prerequisites

When creating a 3D application we have technical and functional requirements that lead us to the choice of a rendering engine. But rendering engines support a limited amount of input formats that makes the choice even more difficult when we also have formats requirements. These requirements could be the need to reuse existing 3D contents or to work with specific modeling tools which formats are not supported. Even if 3D formats and rendering engines have different usages and functionalities, they nonetheless have many similarities. Most of them are indeed based on a scene graph data structure. They also use similar ways to model and organize data in the scene graph. For example they have a similar shape description structure with a separation between geometry and appearance. They also use similar modeling transformations as well as similar primitive shapes. For 3D formats, this can be explained by the fact that since the concept of scene graphs was introduced by Open Inventor [SC92] in 1992, no other concept as efficient and flexible as this one has been proposed. It indeed reflects the underlying rendering pipeline. The explanation for rendering engines is more obvious. In fact they all rely on one or both of the existing low-level API: OpenGL and DirectX. Thus, their evolution follows those API improvements as well as GPUs improvements. It was for example the case when GPUs enabled shader programming.

In [Hin00] and later in [DH02], Döllner and al. have proposed a generalized scene graph structure based on these similarities with a view to improve the rendering process. Steinicke et al. [SRH05b] propose a generic virtual reality software system based on Döllner's work in which rendering can be performed by several low-level rendering APIs. We have used those previous works to design the Scene Graph Adapter API.

3.7.2 Overall architecture

Figure 3.17 depicts our architecture. The purpose of the Scene Graph Adapter API is to enable communication between an input scene graph of a given 3D format and the output scene graph of a rendering component (rendering engine, physics engine, behavior engine, etc.) in a 3D application. This application can be a game, a plugin, a GUI for a 3D display and so on. We call the first scene graph the *format scene graph* and the second scene graph the *engine scene graph* (called the *renderer scene graph* as here we will only use one rendering engine).

To complete our architecture we need components that load, decode and adapt a *format scene graph* using the Scene Graph Adapter API on the input side. Similarly, on the rendering side of our architecture, we need a component that adapts instructions from the Scene Graph Adapter API into instructions of the renderer API to build the *renderer scene graph*. We call these two components the *format wrapper* and the *renderer wrapper* respectively. There is only one format wrapper for every file of a given format as well as one renderer wrapper per rendering engine.

Format wrappers do not depend on an application. They can be reused in any application providing that it relies on the Scene Graph Adapter API. Yet a *Format Wrapper* depends on a Format Decoder. Format Decoders are composed of existing tools like a parser or a loader that help at developing a *format wrapper*. Every 3D format indeed comes with at least a viewer so that this part should not be made from scratch. Besides as every 3D format possess specific features, reusing these pre-existing tools helps at keeping them up to date. The Format Decoder decodes and parses the input file and builds an internal scene graph data structure of the *format scene graph*. Depending on the used decoder, this component may include an update message handler that receives event update messages and updates relevant nodes accordingly. If it is not the case, then it must be implemented in the *format wrapper*. The *format wrapper* then uses this representation to adapt it using methods from the Scene Graph Adapter. Our architecture allows to mix several *format wrappers* within a single application in order to mix different input formats.

Renderer wrappers do not depend on an application either and can be reused in any application based on the Scene Graph Adapter. A *renderer wrapper* relies on the rendering engine API in the

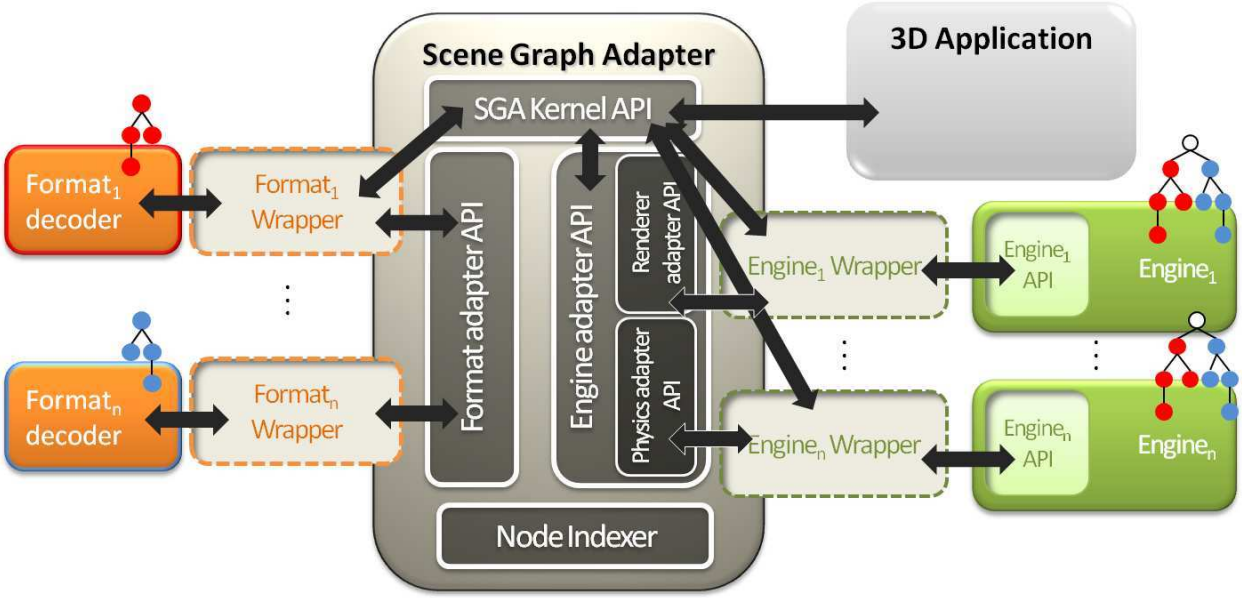


Figure 3.17: Our architecture allows the loading of any 3D graphics format simultaneously in any available rendering engine. The scene graph adapter is an interface that adapts a scene graph (SG) of a given format into a renderer scene graph and which also allows the rendering part to request this scene graph.

same way a *format wrapper* relies on a Format Decoder. It uses methods from the Scene Graph Adapter API and adapts them using the rendering engine API to build and update the *renderer scene graph*. As explained in 3.7.1, the choice of a rendering engine is crucial when designing a 3D application. Therefore it is important to enable the use of most of the available rendering engines.

3.7.3 Benefits

The SGA architecture addresses the reusability issue and achieves the rendering of any 3D format without functionality loss. It has several benefits:

- First, it fully supports all the scene-graph-based 3D formats without rendering limitation, assuming that we have the appropriate *format wrapper*. It makes it possible to use old 3D models without transcoding and functionality loss. Thus it allows a more efficient collaborative work. Research teams can share their resources without being hampered by compatibility problems. Furthermore, depending on the requirements of an application, it is possible to use the most appropriate format without rendering engine restriction. It also makes easier the porting of a format into rendering engines; this can help to promote a new format.

- Second, it works with any rendering engine without input format restriction. Once a *renderer wrapper* is available, it gives access to every available *format wrappers*. An application can use the most appropriate rendering engine without being hampered by format compatibility. Designers team can use any modeling tool and export their 3D models without compatibility problems. It is also possible to directly import a modeler native format in the application while avoiding transcoding drawbacks.

- Third, it makes it possible to mix and reuse *format wrappers* and *renderer wrappers* as required by application. It allows to mix 3D formats and their functionalities. We can for example load a Collada model with physics properties in an X3D world and explore it using

X3D's navigation and interaction features. Besides a wrapper can be reused in any application that is based on the Scene Graph Adapter API. It allows development teams to share their wrappers hence facilitating components reuse and teams cooperation. In addition, more and more 3D application use third-party rendering components instead of creating new ones. There are physics engines (Havok [1], PhysX [2], Open Dynamics [3], etc.), character animation engines (Granny [4], Havok Animation [5], etc.) or AI components (Kynapse [6]). Thus, a wrapper for those components can be reused in many applications.

For more information about the SGA, [BBRDA11] and [BBDRA11] illustrates how the different components are working and how they interact together. The Scene Graph Adapter allows the rendering of multiple 3D formats simultaneously in a single view but also manages interactions and animations declared in the input files. To achieve this, the Scene Graph Adapter API provides a two-way communication between format wrapper and renderer wrapper.

3.8 Conclusion and future work

The PAC-C3D architectural model is an explicit evolution of the PAC model dedicated to 3D collaborative virtual environments. Each shared virtual object of a CVE must be decomposed into three main kinds of components described by three interfaces. The *Abstraction* is in charge of the core data and behavior of the object, the *Presentations* are in charge of the presentation of the object to the user, and the *Control* is in charge of the consistency maintenance between *Abstraction* and *Presentations*, and between all the distributed *Controls* of the shared object.

PAC-C3D can deal with the main distribution modes encountered in CVE and it has been validated with several distribution policies, on local area networks and on wide area networks over the internet. PAC-C3D also makes it possible to design a CVE with very small dependency on a 3D graphics API, and it makes it easy to use different 3D graphics API for different nodes involved in the same collaborative session, providing easy interoperability between 3D graphics API. It is also possible to rapidly couple other 3D engines (for example physics engines) by adding another *Presentation* (related to the engine) to PAC-C3D objects.

The next steps are to take more efficiently into account "active" *Presentations* and to couple PAC-C3D objects with other kinds of engines, for example Artificial Intelligence behavior libraries that could be used in the same way as a physics engine to drive virtual objects.

Although it was not initially designed using the PAC-C3D model, the Scene Graph Adapter architecture is compliant with PAC-C3D and presents another approach to design and implement VR applications with a strong separation between rendering engines. We now have to validate this architecture for CVE, which should be possible by adding a network engine to the SGA architecture, this is a work in progress.

[1] http://www.havok.com/index.php?page=havok-physics
[2] http://www.nvidia.com/object/physx_new.html
[3] http://www.ode.org/
[4] http://www.radgametools.com/granny.html
[5] http://www.havok.com/index.php?page=havok-animation
[6] http://usa.autodesk.com/

Part II

Models for Designing Collaborative Interactions

This part deals with the models that can be used to design interaction and collaboration within CVE. Its main concern is about the abstract level viewpoint upon a CVE architecture, but these models have also to be deployed at the presentation level and they must be driven by the physical input devices of the users, as illustrated in Figure II.1.

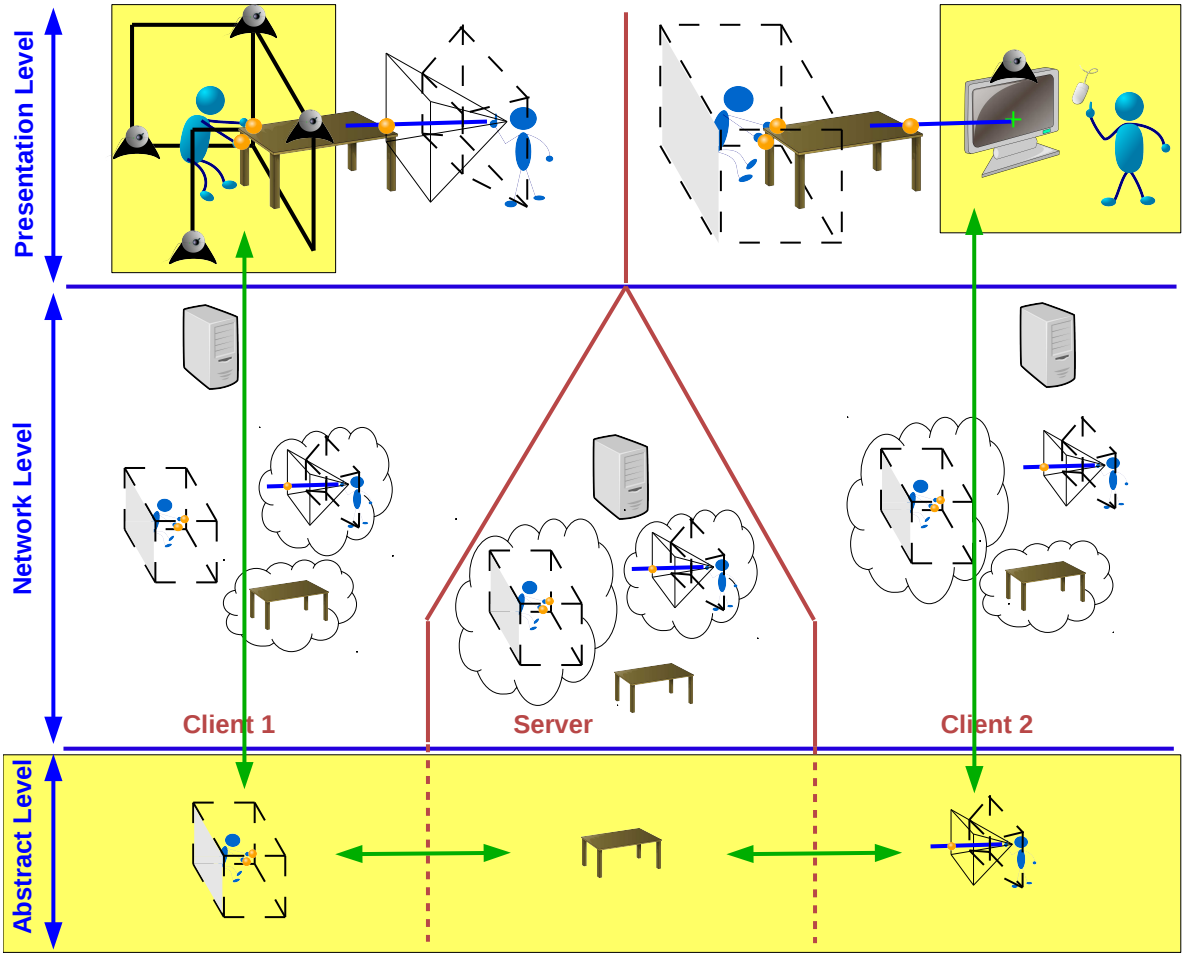


Figure II.1: Design of collaborative interactions at the abstract level of a CVE architecture, with an extraction, from the presentation level, of the features of the users' physical workspaces.

Indeed, building virtual reality applications is still a difficult and time consuming task. Software developers need a common set of 3D widgets, hardware device abstraction and a set of software components that are easy to write and use. These means are intended to provide collaborative interactions in rich virtual applications and easy use of many input devices to drive collaborative interaction metaphors.

In chapter 4 (Modeling Interaction and Collaboration) we propose a solution for expressing interaction and collaboration in 3D CVE, by adding interactive and collaborative features to virtual objects, to model both interactive objects and interaction tools (special virtual objects dedicated to interaction), and to model how these two kinds of virtual objects can communicate together. As a consequence, we describe a communication protocol between interaction tools and interactive objects. We then obtain users on different sites that are able to interact in a shared environment with interactive objects that are provided with access levels.

Then, chapter 5 (Metaphors for Collaborative Interactions) deals with metaphors for interaction within Collaborative Virtual Environments (CVE). Usual 3D interaction metaphors must be adapted to fit with collaborative interactions, and to make users aware of this collaboration. We

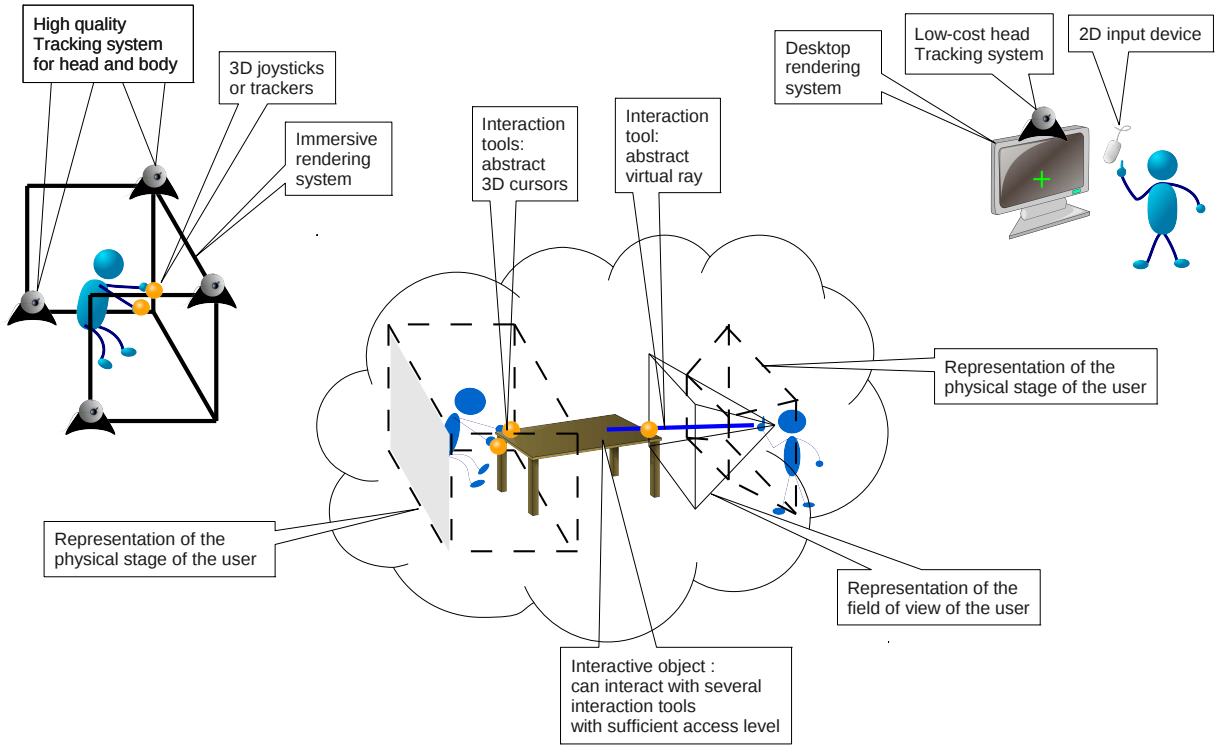


Figure II.2: Modeling collaborative interaction in a CVE.

have contributed to this field through new metaphors for multi-user interaction.

Although it is important to work on abstract interaction and metaphors, indeed sometimes the hardware features of the real environment of the users must be embedded in the run-time VR application to take into account the physical environment of the users, such as the size and the resolution of the screens of a VR immersive system. So we need both to develop VR software that would be loosely-coupled with hardware input and output devices and with rendering software, and which would also be able to adapt itself at run-time to available hardware and software. We need models to achieve this link between the different levels of a CVE architecture detailed in Figure II.2.

In chapter 6 (Modeling Users' Physical Workspaces) we explain why most Virtual Reality (VR) systems must consider the users' physical environment to immerse these users in a virtual world and to make them aware of their interaction capabilities. Actually, no consensus has been found in the existing VR systems to embed the real environment into the virtual one: each system meets its particular requirements according to the devices and interaction techniques used. So, we propose a generic model that enables VR developers to embed the users' physical environment into the Virtual Environment (VE) when designing new applications, especially collaborative ones. The real environment we consider is a multi-sensory space that we propose to represent by a structured hierarchy of 3D workspaces describing the features of the users' physical environment (visual, sound, interaction or motion workspaces). A set of operators enables developers to control these workspaces in order to provide interactive functionalities to end-users. Our model makes it possible to maintain a co-location between the physical workspaces and their representation in the VE. As the virtual world is often larger than these physical workspaces, workspace integration must be maintained even if users navigate or change their scale in the virtual world. Our model is also a way to carry these workspaces in the virtual world if required. It is implemented as a set of reusable modules and it has been used to design and implement multi-scale Collaborative Virtual Environments (msCVE).

Chapter 4

Modeling Interaction and Collaboration

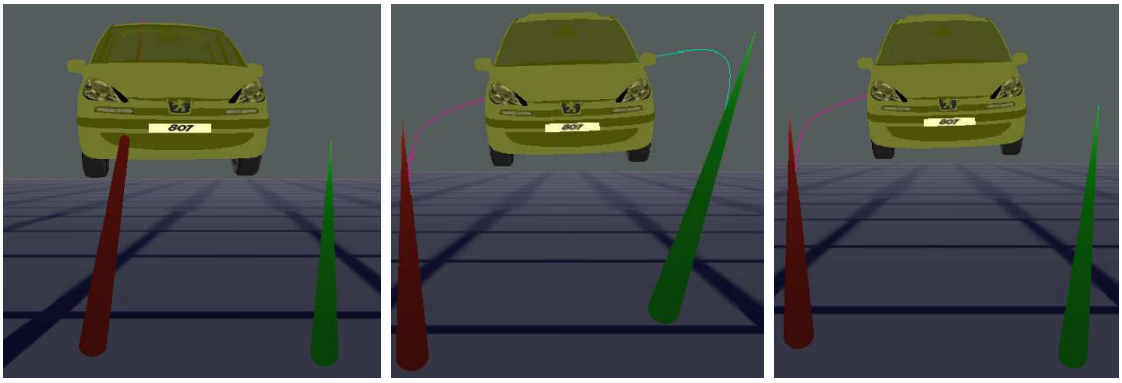


Figure 4.1: At left, only one ray (the one at left) is controlling car position. At center, both rays are bent (see lines coming from them) because they attempt to modify car position at the same time in opposite directions. At right, the right ray left its control but the left ray continues its action. (Car model courtesy of PSA Peugeot Citroën)

4.1 Introduction

From a developer point of view, building virtual reality applications is still difficult as opposed to the development of 2D applications as stated by [BHB08, OF04]. In fact, building 2D applications can be a matter of an assemblage of a set of well known 2D elements in a graphical application. At the same time, many VR platforms still do not propose a common set of 3D widgets but sometimes the connection of low level blocks in an editor (e.g. the Virtools platform). To ease the development of VR applications, many authors describe abstractions of hardware devices (e.g. [OHL+04]). With 2D applications, people generally use a 2D mouse and a keyboard whilst high-level VR applications propose optical markers for 3D tracking, haptic devices, speech recognition, etc. Hardware devices need an abstract code layer to ease their deployment, use and replacement by other hardware devices. Also, the application code is built of this abstract layer in order to not depend directly on a very specific hardware device. This goal has led to the development of many metaphors for interactions (virtual rays [PBWI96, BJH99], virtual hands, Go-Go [PBWI96], etc.) that are driven by the abstract hardware layer. From a user point of view, it is possible to replace a hardware device, for instance a 3D mouse, with another one, for instance a 3D optical

tracker, while still using the same interaction metaphor. Also, interaction tools are equivalent to the pointing or manipulating techniques found with 2D applications. An interactive object is any object in a virtual environment that users can interact with. An interaction tool [BH95] is an object in a virtual environment that will send data to an interactive object; a tool will also receive data from an interactive object in order to track its behavior. Finally, an interaction tool may be driven by hardware devices in order to let a user interact with an interactive object, or an interaction tool may be driven by a software component (e.g. a virtual agent with a behavior). As a consequence, a VR application contains interaction tools and virtual interactive objects.

In this chapter, we propose a communication protocol to describe the messages that an interaction tool and an interactive object will exchange, as described in [ADA09a]. It is an extension of the protocol presented in [DM00a] and in [DLT04]. This protocol provides several advantages. For developers, it will ease the the development of new metaphors. For users, il will enable: natural collaborative interactions at the same time on objects; multi-sites interactions; easy deployment, because our protocol aims to provide VR platform interoperability. As well, abstraction of hardware devices let users use different kinds of devices and change them at runtime depending on tasks.

As we stated, many VR applications are written from scratch instead of reusing existing solutions. Usually, they are written around a 3D engine (such as OpenInventor, OpenSG, Ogre, Java3D or jReality) or even at a lower level around a 3D rendering system (such as OpenGL or JoGL), or around a toolkit such as ARToolkit for augmented reality applications. In addition, applications based on VRML mix up in the same graph several scene parts to describe hierarchies of geometric models and behaviors/constraint of the interactive elements.

We propose to use a set of software components that have to be aggregated together. The dependencies between these elements are described through the connection of these elements in a configuration file. These extensions have been implemented and tested with OpenMASK [LCAA08] but can be implemented on other platforms. Here we will describe communications between interaction tools and interactive objects, and how they are implemented.

In this chapter, section 4.2 presents related work, focusing on hardware device abstraction, interaction tools programming and reusability. Section 4.3 compares our approach to previous methods. In section 4.4, we describe our model: interaction tools, interactive objects and how they are linked together. In section 4.5, we describe the communication protocol with different cases of interaction, and section 4.6 presents our proposition for describing interaction and collaboration in a declarative way. Section 4.7 presents another approach, based on 3D files composition, for describing interaction between virtual objects described using different file formats. Finally, we conclude and then we give some cues leading to future work.

4.2 Related work

This section presents related work for interactions with VR platforms. The first part deals with hardware device abstraction which enables the development of many interaction metaphors that we present after. Finally, we also explain that awareness is required to help users or the system to make collaborative interactions possible.

4.2.1 Hardware device abstraction

The large variety of hardware devices in AR and VR makes them hard to use for software developers. Previous work has undergone to provide a set of basic components to obtain platform-independent code that can speed up the creation of new VR platforms [BLO+05]. DWARF [ISA01] is a framework of reusable distributed services described through XML. For instance, the service manager will connect a service providing a captured picture, thanks to a camera, to a system analyzing the

picture for tracking. Services managers running at each system site communicate with CORBA. MORGAN [OHL+04, BLO+05] is a framework for AR/VR that classifies hardware devices within a hierarchy. An hardware device is then derived of a set of classes. For instance, a mouse is a child of the classes MousePointer, Button and Wheel. Figueroa et al. [FGW01] present a software architecture using filters connected in a dataflow. A filter that read data from an hardware device encapsulate code to acts as an hardware driver. For example, there might be a 2D mouse filter. Also, other filters can be connected to its outputs to read the data it sends. OpenTracker [RS01] is also based on a dataflow mechanism where filters are here named nodes. A set of connected nodes forms interaction tools as stated by [FGW01]. Each node is made up of one to many inputs whereas it has only one output. Inputs and outputs are both typed and OpenTracker allows a connection from an output to an input only if they are of the same type.

4.2.2 Interaction metaphors

Interaction tools depend on the abstracted hardware layer and are involved in metaphors commonly found in VR applications. A taxonomy of the main interaction metaphors is provided in [BKLP04].

Implementation

Metaphors usually depend on available devices as stated by [BHB08] when no abstract layer is provided for hardware. However, many of them, like hand metaphors (e.g. "Go-Go" [PBWI96]) and pointer metaphors (e.g. ray-casting [PBWI96, BJH99]), need to obtain information about the interactive object they want to interact with, for example the object's position. Figueroa et al. [FGW01] use a set of inputs and outputs for data, but they seem to hard-code connections between tools and interactive objects and to limit interactions to the control of the object's positions.

Instantiation

As a result of hardware device abstraction, many higher-level interaction paradigms have been implemented. With dataflow based platforms, they are composed of a set of connected components (e.g. filters or nodes). A graphical system can be used to connect the objects of a scene. Unit [OF04] lets a user compose its tools via the connection of their properties. InTml [FGW01] relies on a set of interconnected properties and define a XML format, as an X3D extension, to describe 3D interactions. CONTIGRA [FGH02] proposes also an XML-based format, as a X3D extension, based on its own scenegraph. Finally, authoring tools has been developed, which can use for instance a XML-based format to store produced virtual worlds, to hide low-level/technical issues.

4.2.3 Feedback to the user

From the interactive objects' side, Smart Objects [KT98] encapsulates within the object descriptions of its characteristics, properties, behaviors and the scripts with each associated interaction [SVP07]. With such an approach, an interactive object can give useful information to the interaction system in order to provide helpful feedback to the end-user. Many authors demonstrated that feedback is required to help users. Firstly a user needs to understand the actions he is able to apply to an interactive object: object position update, orientation update, color update, scale transformation, etc. Visual metaphors using arrows or cursor indicate that positions can be updated. These visual metaphors can then be used during interaction and be merged with other modalities [SLMA06]. Feedback helps a user to understand what his action causes. In a collaborative virtual environment, users must also understand the actions done by other users to help them, to continue theirs actions or to do a simultaneous action [GG99]. The following section compares our approach with the existing literature about behavior coding, feedback to the user / awareness coding and VR software architecture.

4.3 Enabling interaction between interaction tools and 3D objects in CVE

We want to enable distributed interactive sessions with multi-users at different sites but sharing the same virtual environment. Each user will interact with interactive objects through interaction tools. Interaction tools and interactive objects are equivalent to filters [FGH02], or nodes [RS01]. We name them virtual objects. Our platform, OpenMASK [LCAA08] manages low-level communication between them. We propose a communication protocol for these objects in order to:

- Ease development of metaphors: for an interaction tool, it is not needed to develop a specific code for interaction with an interactive object of type "A", then code for an object of type "B", etc. Dialog between interaction tools and interactive objects is normalized.
- Change dynamically access permissions to interactive objects at run-time and/or before in a configuration file.
- Provide more dynamicity to interactive objects: a property can be added or removed dynamically. For instance, an interactive object may be static because it does not offer a position attribute. But this interactive object will become movable if it adds such an attribute after it received a particular event for instance.
- Ease deployment of virtual reality platforms: the VR platform does not need to be programmed in a specific language because our protocol defines its own introspection mechanism and does not need to use technologies such as Java remote method invocation. Our platform is currently implemented in C++. Moreover, using an introspection mechanism avoids us to need a distributed scene-graph which describes properties, unlike [SH02]; it avoids also to use complex systems like distributed shared memory models [Tra99] to spread properties.
- Provide interoperability between different VR platforms at run-time: a language neutral system is easier to port to many programming languages.
- Propose a new communication protocol for communication between interaction tools and interactive objects. In fact this protocol, while more adapted to distributed applications, works also on a single host. This protocol introduces a small overhead to the connection phase since a tool and an interactive object have to become interconnected. When a property of a tool has been connected to the property of an interactive object, the tool sends data as it would if the connection was hard-coded.

We have implemented the protocol communication with virtual objects which are used as empty shells that encapsulate software extensions. A software extension is a software component that needs a virtual object as a host. Each of them will be executed sequentially at run-time by its virtual object. Once software extensions have been assembled in an interaction tool, VR application designers can therefore use it instead of a set of many small virtual objects and speed up building of VR applications. For VR application developers, extensions introduce many advantages:

- Improving software engineering: the models proposed in [FGW01, RS01] lead to the creation of for-general-use filters/nodes. For instance, if you need a very specific filter for an hardware device, you will create a filter of type *SpecificFilterForMyDevice* and it will look like any general filter or node while it is not. Software extensions clearly describe pieces of code that are aimed to be used inside a main structure (i.e. a virtual object).
- Improving performances at run-time: our platform is multi-sites, in which case we can improve performances if objects are correctly distributed. So, extensions can help us toward this goal

because they lead to a unique data structure to distribute, a virtual object, rather than a set of many, filters/nodes. Extensions migrate automatically with their virtual object.

- Improving scalability/adding dynamicity: it may be interesting to avoid adding heavy and complex behavior on some virtual object all the time, to lower CPU or network usage. Therefore, a mechanism such as our software extensions, introducing a way to add or remove software components at run-time, is required. Let us take a very simple filter that will send positions to move an object up and down. To change this behavior, another filter, for instance a filter to move from left to right, will be required. First, it will be need instantiate this filter, if needed, then to disconnect the up-and-down filter, connect the object to the left-to-right filter and eventually remove the useless up-and-down filter. Moreover, creation, destruction, connection and disconnection will have to be managed by a supervisor, so another filter. With software extensions, one object will manage itself its up-and-down extension, after it will have received an event for instance, with a left-to-right extension.

4.4 Model: tools and interactive objects

We define an interaction as a bidirectional communication between a tool and an interactive object: a tool sends commands to an interactive object that is in charge of treating the commands, and it also receives commands from interactive objects.

4.4.1 Tools

We assume that an interactive object is only manipulable through a tool. Our assumption leads to a situation where every user who wants to interact with an interactive object will have to use a tool. A user will interact with an object through a tool: a hand, a ray, a pointer, etc. This user can be a person or a software component.

A tool modifies properties of an interactive object so it is made to send data to an interactive object that will treat these data. A tool is made up of a set of attributes where each of them has a type and an access level. They define data that a tool will be able to send. For instance, a ray intended to move an object would need to embed a position attribute. The position attribute of the tool contains the position value and is sent to the interactive object when the value is updated.

From the tool's point of view, the communication flow with an interactive object follows three steps: i) initialization, ii) use, iii) release of resources. Those three steps fit with how an interaction happens: i) selection of an object, ii) manipulation, iii) object release. At selection, a tool starts a particular communication with an object designated by a user with a tool (e.g. a ray, a menu, etc.). Then the tool requests the possibilities of interaction that the interactive object offers and presents them to the user. The user can now start manipulation on some object properties: we call this state the manipulation state. The tool is now sending values to the interactive object (e.g. a new position, a color). If many tools are sending values to an interactive object, we say that those tools are interacting cooperatively and so the interactive object will have to deal with those concurrent inputs, we will explain this case in the following section.

4.4.2 Interactive objects

An interactive object must give some knowledge to tools about which control it offers. Our model assumes that each interactive object embeds a set of attributes which define the data the object can receive. An interactive object can be interrogated to give its interaction capabilities.

When a tool knows the capabilities an interactive object offers, it can take control of some of the attributes to modify the data they contain. As a consequence, any tool can modify any properties of an object. We define a control interaction access policy for the tools in order to regulate which

of them can interact. An interactive object arbitrates actions from the tools: it can accept or refuse any action from them. Also a tool does not have any way to directly stop interaction of a concurrent tool. Moreover, a control can be interrupted. This access policy is associated with attributes of an interactive object.

Each attribute has a type, a shareable flag, an informative entry and an access level. The type and the informative entry are used to match a tool attribute with an attribute of an interactive object when a tool tries to take control of an interactive object attribute. For example, we need to make a tool aware that the object it wants to control position has a position attribute, and we also need to connect the tool to the position attribute.

A shareable attribute must be able to deal with concurrent actions. In this way an attribute is preceded by a converter. A converter aims to handle concurrent actions and convert them into a value that the associated attribute will use as a new value to contain. A converter may compute the mean value of a set of positions. We do not propose any way to handle contradictory actions like a Boolean set to true by a tool while another tool sets it to false. Contradictory actions will usually lead to the last action received being the result stored into the current attribute.

Finally, some tools can be manipulated as interactive objects to move them or change their color for instance, so those tools are both tools and interactive objects at the same time.

4.4.3 Relations between tools and interactive objects

Figure 4.2 gives the general picture of how two tools control the position of an interactive object. An interactor module is embedded in a tool and has to compute a new position to send to the interactive object. The interactor module must also listen for events coming from the controlled interactive object to track its activity: if a new tool is connecting to the interactive object for instance. Finally, the interactor module implements the tool part of our communication system. An interactive object embeds an interactive module which implements the interactive object part of the communication system. The fusion converter presents an attribute to let the two tools modify the object position. For positions, it can be limited to the computation of average values positions from n values that it received.

The interactor module and the interactive module are both a piece of software that can be added or removed dynamically to any object to turn it into an interactor or an interactive object. We will see that they contain software extensions.

4.5 How to make interaction tools and interactive objects communicate?

This section explains how one interactive object can be manipulated by many tools, then how a tool can manipulate one or many interactive objects.

4.5.1 One interactive object with many tools

We consider the two ways an interactive object can be manipulated: first only one tool is manipulating the object; second at least two tools are manipulating the (same) interactive object at the same time.

Simple control

The sequence we describe in this section is illustrated in Figure 4.3 and shows one tool that will control all the attributes found as accessible in an interactive object. First, the tool opens a session

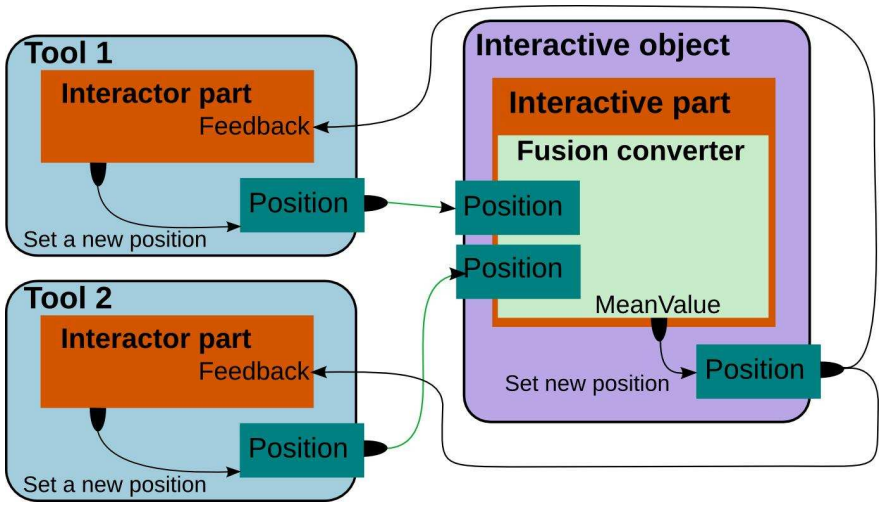


Figure 4.2: Global organization of two tools controlling position of an interactive object.

with the interactive object previously selected by the interactor, which can be a human person or a software component.

A person may select an object via ray-casting for instance. Then the tool needs to interrogate the interactive object to obtain the object attributes it would be able to manipulate:

- It sends a *get_accessible_parameters* command to the interactive object.
- The interactive object answers with an *accessible_parameters* message which is also made up of a list of IDs of accessible parameters of this object for this tool.

This interrogation system does not rely on an introspection mechanism given by a programming language such as Java, but it defines a set of types given by a configuration file for the virtual environment and associates some meta-data to describe the attributes. This way we define a programming language agnostic protocol. Now the tool can try to take control of some attributes and later to send new values to the interactive object:

- The tool attempts to take control of the accessible parameters. For instance, if it tries to take control of a position attribute, it will send a *control_take_over_and_get_current_values* message with an ID for the position.
- The interactive object uses a *current_value* message to send back a value for the position; this message is composed of an ID for the position attribute and the position value. When all the values the tools were interested in have been sent then the interactive object sends a *control_taken_of* message, which is composed of the IDs that the tool is controlling. This message acts also as an end flag. Now depending on the values the tool received, it is able to initialize itself for further computations. With a position, a tool can compute an offset between itself and an interactive object to simulate a fixed slider constraint when it moves the object.
- The tool can now propose new values to the interactive object with a *send_value message*. The tool will send as many messages as they are values to send.
- The interactive object periodically informs a controlling tool of the values of the attributes the tool is controlling with an *effective_value* message. Each message contains the value of a controlled attribute.

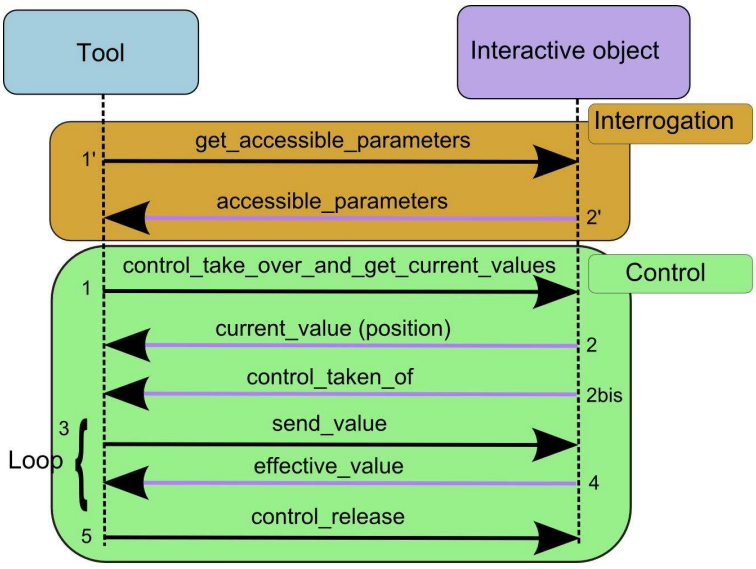


Figure 4.3: Communication sequence between a tool and an interactive object.

- The tool can release control of an attribute it is controlling with a *control_release* message. This message is sent with IDs of the attributes to release control of.

Shared control

A shared control stands for a control where at least two tools are manipulating the same interactive object. At this time each tool needs: i) to know the other tools that are manipulating the same object, ii) to track the object's evolutions to inform its users (human person or software component) about incoming/outcoming of tools, iii) or to adapt its internal computations. A human user adapts his actions depending on the actions that other tools are doing: for example, in order to help another user, someone needs to become coordinated with the user or the actions he made, etc. Let us consider a tool named T1 which is interacting with an interactive object named O. If another tool, named T2 takes control of some attributes of O then T1 will receive a *control_taken_by* message made up of the ID of T2 and also the IDs of the controlled attributes. When T2 releases the control of some attributes, T1 will receive a *control_released_by* message made up of the same kind of attributes a *control_taken_by* message uses. Figure 4.1 illustrates how two rays manipulated by two human people can move a car together. A ray becomes bent when the position it tries to apply is constrained by, for instance, the position given by another tool. This metaphor is described in [DF02]. The use of the *control_take_over_and_get_current_values* message is not decomposed into two commands to avoid a mutual exclusion. Concurrency is supported by the interactive object itself to let many people interact together. Moreover, if we first ask for control and latter for values, we will introduce an important delay between object picking and the beginning of object manipulation. This is a main issue when a user wants to pick a moving interactive object.

4.5.2 One tool with many interactive objects

We consider that a tool can manipulate an object if it has an interactive extension. We will explain how one tool interacts with one interactive object. Then, we will explain what a complex interactive object is and how one tool interacts with.

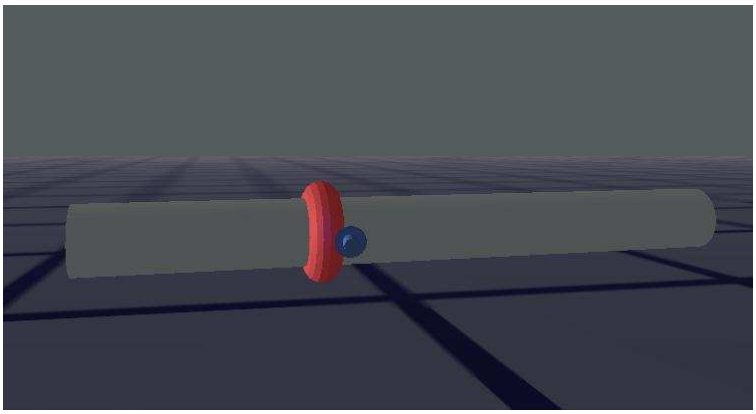


Figure 4.4: This 3D widget is a slider which is made up of a ring sliding along a bar. Here the ring is selected and we can see a small pointer in front of the ring that a user can manipulate to move the ring.

Control of a single simple interactive object

This case is the one presented in section 5.1.1. Also, a tool takes control of the set of attributes it asked for. After the manipulation of the position of the interactive object, the tool releases its control.

Control of a single complex interactive object

We qualify an object as complex when it has a "big" number of attributes to interact with. Such an object requires a particular approach for interaction. Some virtual objects would require a composition of interactive objects: a "virtual" car may be made up of doors, wheels, an engine, a chassis, etc. where each of these pieces embeds an interactive extension. Some constraints would be added to interactive objects thanks to software extensions in order to keep doors attached to the car; maintain seats within the car, etc. so an interactor can move the entire car and interact with some of its parts without disassembling everything. The interactive feature is described here through a local approach.

Let us consider a flexible hose made up of many rings so it may be hard to add local constraints between them and compute positions. A global approach may be needed: only one (or at least a small number) interactive object is employed to implement a finite element method. When an interactor wants to apply some actions on the flexible hose, it has to go through the unique interactive object which computes all the properties of each ring. This feature needs to be able to decompose a flexible hole into a set of rings selectable by the user. Associations between rings and attributes of the hose are described in a configuration file.

A user may select an object part by different ways: a menu, voice, etc. As an example, we assume that a user selects an object by ray-casting. A 3D mesh is decomposed into a set of 3D submeshes so a user will pick one of them. Each 3D submesh is associated with a given ID thanks to the configuration file describing the virtual environment. Now, when a user picks a 3D submesh it obtains an ID that it sends through a get_accessible_parameters message. The interactive object receives an ID of one of its part, and then it answers with the associated attributes. This message behaves like a filter for the interactive object attributes.

As a simple example, Figure 4.4 shows a 3D widget made up of two parts. At screen, there are two 3D meshes (a ring and a slide) but only one interactive object allows interaction and it keeps these two objects together. If the interactor selects the ring then it will move it along the slider, if

it selects the rest of the slider it will move all the slider including the ring.

Control of a set of interactive objects

With a set of interactive objects, a tool does not directly interact with interactive objects but through proxy-tools (see Figure 4.5). A proxy-tool is an intermediate tool between the "real" tool, that the user manipulates, and the interactive object. Note that to simplify, we did not decompose tools in a main tool + proxy-tools in section 5. The use of a proxy-tool aims to add more flexibility to how a tool is programmed: the tool does not have to contain code to communicate directly with one interactive object or many. A proxy-tool embeds the code to manipulate an object: a control extension and a behavior extension. Also it becomes easier to manipulate many interactive objects because the tool only has to instantiate many proxy-tools and then follows their evolution.

Moreover a proxy-tool can have its own extensions that are different with the tool. Those extensions can add a specific behavior to each proxy-tool. In addition, proxy-tools can be spread on the network to reduce latencies and network traffic during a distributed session.

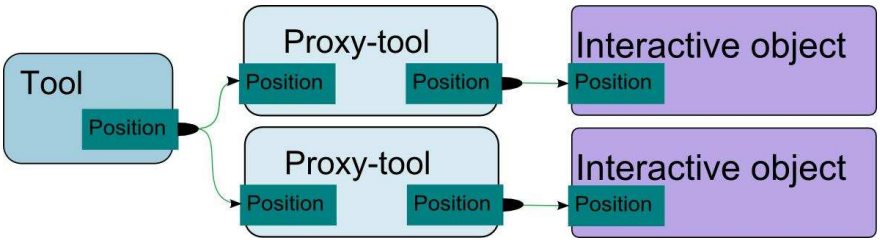


Figure 4.5: A tool uses proxy-tools to manipulate some interactive objects. The main tool sends its position then each proxy-tool computes a position for an interactive object using the main tool's position.

4.5.3 Access to interactive object properties

Platforms for virtual reality may be used to simulate real-life interactions where a trainer explains industrial maintenance procedures to technicians, a boss accesses to confidential data, etc. Prior work has been done in [GPS00], for instance, but its policies seem quite limited.

Each attribute of an interactive object is provided with an access level. When a tool tries to manipulate an attribute, it first needs to have an attribute that matches the expected interactive object attribute: same type, corresponding meta-data and a sufficient access level. This policy leads to the fact that a tool can fail to take control of some attributes, or it can be "kicked out" by another one. When a tool is interacting with an interactive object, another one can come and raise access level to a higher value, thus the former tool lost its control and receives a control_ended message which is made up of IDs of the released attributes. In order to implement this policy, we propose to manage interactive properties that can be associated with interactive objects themselves or interactive object attributes. Let us describe elements of such a property:

- A name, a type (e.g. Integer, Position, Orientation, etc.), a meaning (i.e. a kind of comment), an association with a feature of the interactive object (e.g. interactive object position) and the number of tools that can interact with it at the same time.
- A priority level in order to determine the lowest priority needed by an interaction tool to be allowed to interact with an object. This priority can be set globally for all tools, or for each existing group of tools, or for an object. The priority can be set globally for an object or specifically for each of its attributes.

- An access policy for an interactive object attribute or a group of interactive objects. It determines how the interactions can be shared between several tools. The different policies allow either many interaction tools to share interactions with an interactive object, or only one tool to interact with the object. Currently, we propose three policies:
  - "Unfreezable" policy: a tool interacting with such an attribute will not be able to forbid another authorized tool to stop or join the current interaction. Motivation: a tool will not be "kicked out" by another one.
  - "Freezable at any level" policy: a tool interacting with such an attribute will be allowed to determine, for each existing group of tools, the minimum required level needed for a tool in order to be able to stop or join the current interaction. Motivation: a tool will give access to only a set of tools.
  - "Freezable at tool level" policy: nearly the same policy than "Freezable at any level" but the level cannot be superior to the priority of the tool that initiated the interaction session. Motivation: a tool will lower the access level to allow new tools. For instance, we imagine a powerful user that give access to some people by lowering access level.
- A description of how the feedback used to make the user aware of its interaction (with an attribute, an object or a group of objects), have to be triggered: before, at the beginning, during or after the selection and/or the manipulation of an interactive object or attribute.

4.6 Description of interactive and collaborative properties

All the properties defined in section 4.5.3 can be declared in configuration files that make it possible to describe all the interactive and collaborative properties of the content of a virtual environment.

A first implementation is available for the OpenMASK platform, allowing to describe both interaction tools and interactive objects.

```
<partage:interactive_model id="mon_objet" target="mon_objet">
  <partage:interactive_access mode="ABSOLUTE" count="0"
                              policy="Freezable at tool level">
    <partage:group_control group="local" level="2" />
    <partage:group_control group="invite" level="1" />
    <partage:group_control group="inspecteur" level="3" />
    <partage:group_control group="*" level="0" />
  </partage:interactive_access>
  <partage:interactive_object_property
     name="mon_objet_Interact_TranslationXYZ"
     type="translation" type_meaning="translation_XYZ"
     target="mon_objet/translation" />
</partage:interactive_model>
```

Figure 4.6: Collada description of the interactive properties of a virtual object.

Another implementation has been made as en extension to the Collada format. A new profile (named *partage*) has been created to describe these interactive properties through two new libraries:

- *library_interactive_scenes*, which contains the *interactive_scene* tags, and that is instantiated once to describe the virtual environment, this instance can contain several instances of *interactive_model*;

- *library_interactive_models*, which contains *interactive_model* tags.

Figure 4.6 shows a brief Collada description of an interactive object. More details can be found in [Agu10]. Here is a short description of the new tags:

- *interactive_access*: defines the access restrictions to the attributes of an object.
- *group_control*: for each group, defines the level required for an interaction toll to be allowed to access to the attributes of the object.
- *interactive_object_property*: defines the properties of the interactive object.
- *interactive_feedback*: can initiate a feedback toward interaction tools.

4.7 3DFC: a new container for 3D file formats compositing

We have also worked upon a new formalism to mix several 3D files format and to make them interoperate. It is a 3D container model that enables to compose 3D file formats. It allows not only to compose 3D scenes made of several 3D files of different types but also to combine their features and to make them interact together in the rendering window. This model, called 3DFC for 3D File Container, relies on the Scene Graph Adapter (SGA) architecture presented section 3.7. The SGA makes it possible to load any scene-graph-based 3D file format in a 3D application whatever the involved engines (rendering engine, physics engine, etc.).

The proposed approach consists in defining these interactions inside a container file and describing the 3D files associated and their connections. Our intention is not to propose a new 3D file format but rather to outline a model of container that could possibly be inserted into existing standard formats. We opt for an XML-based model because it is a widespread open-standard, supported by a large number of tools. Our model has been designed as compact as possible, only defining the main nodes required for composition. Therefore it contains a root node (SGA-3DContainer), a grouping node (*Group*), a positioning node (*Transform*) and a content node (*Content*). The *Group* and *Transform* nodes are typical nodes for organizing the scene. The *Content* node allows us reference an external 3D file whatever its format.

4.7.1 Interaction nodes to mix 3D files functionalities

This format allows us to identify and annotate the 3D files that are combined in a 3D scene. We define interaction between these files through three new nodes: a *Route* node, a *Match* node and a *Converter* node.

The *Route* node creates a path between two equivalent properties of two nodes from two different format scene graphs. By equivalent properties we mean properties that have the same semantic. For example the field translation of an X3D transform node and the translate attribute of a node from a visual scene in Collada. The route node contains 7 attributes: *fromFile*, *toFile*, *fromNode*, *toNode*, *fromField*, *toField* and *converter*. *fromFile* and *toFile* values are identifiers of the two format scene graph connected together. These identifiers are defined in the *DEF* attribute of the content node referencing each file. The attributes *fromNode*, *fromField*, *toNode* and *toField* are names of the nodes and fields in their respective format scene graph that are connected through the *Route* node.

The match node allows to add a property extracted from a given node to another node. We can use it to add a mass to a node encoded with a format that does not take into account physics properties. The match node has five attributes: *fromFile*, *toFile*, *fromNode*, *toNode* (identical to those of *Route* node) and *field* (the name of the property to match as it appears in the original format).

The sample file in Figure 4.7 declares a *Route* and a *Match* between nodes of two different format scene graphs.

```
<?xml version="1.0" encoding="UTF-8"?>

<SGA_3DContainer>
  <Group DEF="MyGroupNode">
    <Content DEF="File_1" type="t1"
             decoder_url="" wrapper_url="" url="file1.t1" />
    <Transform DEF='TRANS' translation='-80 60 50' scale="2 2 2" >
      <Content DEF="File_2" type="t2"
               decoder_url="" wrapper_url="" url="file2.t2" />
    </Transform>
  </Group>
  <Match fromFile="File_1" fromNode="Sphere_1"
         toFile="File_2" toNode="Sphere_2" field="mass"/>
  <Route fromFile="File_2" fromNode="Sphere_2" fromField="rotation"
         toFile="File_1" toNode="Cube_1" toField="rotate"
         conversion="converter_1" />
  <Converter DEF="converter_1" type="VectRotToQuaternion" />
</SGA_3DContainer>
```

Figure 4.7: Example of a 3DFC file with interactions between nodes of different 3D files.

4.7.2 Possible uses and benefits of a container

Using a container makes 3D file compositing possible without format compatibility constraints and also allows their contents to communicate. Along with the use of the SGA framework, we have an interoperability model that answers the issue raised by the multiplicity of 3D formats.

The *Route* node is used to copy a property of a given node to another node from a different format scene graph. The properties are shared among different files possibly encoded in different formats.

With the *Route* node, it is possible to organize and break down in our scene among several files in order to improve the reusability of the different involved contents. Moreover, the routing functionality allows us to bring animation in 3D file format that do not provide it. Indeed we have two options to animate an ordinary node: 1) by creating a route between an external animated node and this node, 2) by creating a route between an external animation node and this node. Thanks to the *Converter* node, routes can be created between nodes without being hampered by an incompatibility of units between two semantically identical properties.

The *Match* node is used to match a property of a given node to another node declared in a format that does not offer this property. Through the *Match* node we are able to add properties to a model of any 3D format without altering the original file. The new property is defined by the file format where the match declaration comes from and it is interpreted by the SGA as if the modified node was a node of that format.

4.7.3 Implementation

We have implemented our architecture with three format wrappers and two engine wrappers. The three format wrappers are for X3D, Collada and 3DFC. The two engine wrappers are for Ogre3D and Bullet. More details can be found in [BBRDA12].

4.7.4 Conclusion

The 3DFC solution solves the interoperability issue among 3D file formats. It enables communication between 3D assets encoded with different 3D formats. Indeed through a container file, we are able to describe a 3D scene composed of several 3D contents no matter of their original format and also to define interactions between them.

A container format allows us to capitalize on all features of a 3D format, mix them with other 3D format features and obtain a rich 3D scene without any preliminary alteration of the involved 3D files. In order to use a new 3D format through the SGA architecture, the only thing to do consists in finding a decoder and implementing a format wrapper according to an API given by the SGA. This allows anyone to promote its own file format, whether a small or a big one, and makes it interact with other 3D file formats and 3D application.

The mechanisms offered by 3DFC are also another way to add interactive capabilities to 3D contents without altering them.

4.8 Conclusion and future work

Our motivation is to help software developers to produce rich virtual reality applications, with hardware input abstraction device, software component reuse, easy modification of instances of virtual applications. As a result, we propose a new formalism for 3D interactions in virtual environments. This formalism defines what a virtual interactive object and an interaction tool are, and how these two kinds of objects can communicate together.

We have shown how this communication system works for simple interactions: one tool interacting with one interactive object; and more complex collaborative interactions: two tools interacting simultaneously with the same interactive object, making the users aware of this closely coupled collaboration.

For software reuse, we have developed many software components, which are called software extensions, dedicated to selection and manipulation. The manipulation extension also uses a control extension which implements the communication protocol we described. Interactive objects embed an interactive extension. Thereby tools and interactive objects are easier to implement and describe.

The communication system and the description of what a tool and what an interactive object are, are a step toward a more sophisticated language to describe interactions in virtual reality applications. The description we propose is based on the use of many software components which are assembled.

We also worked on the mixing of interactive features of existing 3D file formats, which is a complementary approach. Furthermore we also think that the union of these format extensions and the description of the communication dialog for tools and interactive objects can enable a better interoperability between different virtual reality platforms.

The formalism we introduced by proposing new extensions to Collada is a first step toward a complete description language to describe the interactive and collaborative properties of virtual objects. Future work could first focus on proposing to make these extensions official for Collada, and to embed them also into other 3D description format standards such as X3D.

We should also embed our interactive extensions in 3DFC, which would be another way to make virtual objects (described for example through Collada or X3D files) interactive through interactive features described at the 3DFC level.

A next step would be also to provide a high-level description of interaction with physical properties of virtual objects, in order to make possible an easy description of more realistic interactions with non-rigid objects.

Chapter 5

Metaphors for Collaborative Interactions

5.1 Introduction

Object manipulation is one of the most fundamental tasks of 3D interaction in Virtual Reality (VR), and many efficient interaction techniques have been developed in this area in the past decade [BKLP04]. Due to new 3D input devices becoming widely available even for the general public, research in new 3D user interfaces is more relevant than ever [BCF+08]. Furthermore, the collaborative manipulation of virtual objects by multiple users is a very promising area for Collaborative Virtual Environments (CVE) [BGRP01]. Collaborative manipulation of objects seems indeed necessary in many different applications of VR such as virtual prototyping, training simulations or assembly and maintenance simulations [RSJ02, DRC+00, FN98, LGH98, SCF97]. In such virtual collaborative tasks, all the users should participate naturally and efficiently to the motion applied to the object manipulated in the VE.

Although most collaborative systems support simultaneous manipulation of different objects by different users, generally only one user at a time can manipulate a virtual object. Interaction metaphors that are usually used for single-user 3D interaction, such as virtual hands, virtual rays or virtual 3D cursors, have to be adapted for collaborative 3D virtual manipulations.

It is important for people sharing a CVE to be aware of the activity of other users, as explained in [FBHH99], in order to help them to understand the evolution of the CVE and to collaborate more efficiently with the other users. Showing the activity of a user to the other users whom he may collaborate with, is a central point for an efficient collaboration, and a lot of work has been realized in this area [FGV+00, GG98]. Many egocentric metaphors, such as the virtual ray casting [PWBI98], are well suited for interaction within CVE, thanks to their graphical visualization that can be shown to the other users.

Therefore, in this chapter, in section 5.2 we make a short state of the art about 3D interaction mataphors and their adequation to CVE. A more detailed state of the art about 3D collaborative interactions can be found in [ADA09b]. Then we introduce some of our contributions to multi-user interaction through the proposition of new collaborative interaction metaphors. Section 5.3 presents our first contribution to simultaneous multi-user manipulation of the position and orientation of a virtual object through deformable rays used ad interaction tools. Section 5.4 presents an asymmetric 2D Pointer / 3D Ray that is as easy to use as a simple pointer driven by a 2D input device, but which provides a 3D representation in order to make the other users aware of the interactions driven by this interaction tool. Section 5.5 recalls our first contribution to multi-point manipulation of virtual objects, by grabbing only two points of a virtual object. Section 5.6 proposes a new 3D interaction technique for 6 DOF multi-user collaborative manipulation of 3D

objects through only positions of three non-aligned manipulation points on this object. Section 5.7 presents a novel concept of reconfigurable tangible interface that can match many shapes of 3D objects, for the purpose of object manipulation by single or multiple users in virtual environments.

5.2 Related work

5.2.1 Two-hand object manipulation

Several 3D interaction techniques have been proposed to manipulate virtual objects with the two hands of a single user [HPPK98]. But only a few of them, such as "grab-and-carry", "grab-and-twirl" and "trackball" techniques [CFH97], enable users to position and rotate virtual objects.

The "grab-and-carry" technique [CFH97] is a 5 Degrees-of-Freedom (DoF) bimanual symmetric tool that enables users to carry and turn an object around with both hands. Object roll is not supported since it is not possible to determine rotation around the axis defined by the positions of the two hands. The "grab-and-twirl" technique extends the "grab-and-carry" technique, adding the sixth DoF using either the left hand's roll, the right hand's roll, or a combination of both. The "trackball" technique is a bimanual asymmetric tool that enables users to use the non-dominant hand to position a virtual object while the dominant hand rotates this object around its center.

In our opinion, these techniques are not very representative of real world interactions considering users' movements. In addition, they could probably not be used to simulate interactions with large or cumbersome objects that a user cannot manipulate alone.

5.2.2 Multi-user object manipulation

Several approaches are suitable to combine two users' movements to obtain the final movement of a virtual object [RSJ02]. A first approach consists in adding the two motions (asymmetric integration of movements). A second approach is to average the two motions. A third approach aims at keeping only the common part (intersection) of the two motions (symmetric integration of movements). But none of these combinations seems ideal. Indeed, the intersection technique is the more relevant when the two users have to perform a very similar action, whilst the average technique is preferred when users have to perform different tasks [RSJ02].

The Bent-Pick-Ray [RHWF06] metaphor enables several users to simultaneously co-manipulate a virtual object. This technique merges users' inputs according to the amount of hand movement a user does with one input device. Rotations are computed with a spherical linear interpolation (Slerp) [Sho85], while the translations are interpolated using only offset transformations. Results may be close to those of the average technique.

The SkeweR technique enables multiple users to simultaneously grab any part of a virtual object through special points called "crushing points" [DLT06]. To determine the translation and the rotation of a grabbed object, the SkeweR considers positions of those points. A problem remains for determining the rotation along the axis linking the two crushing points. The SkeweR will be described section 5.5. A similar technique seems to be used to construct a virtual gazebo [RWOS03]. Two users manipulate a beam by grabbing its extremities. But no solution is proposed for the sixth DoF. This beam manipulation has been reproduced by using two virtual hands but simply using their average position in order to provide a position for the manipulated virtual beam [GMMG08]. In [SJF09], Salzmann *et al.* use two optical markers to let two users manipulate a windshield. Authors also use the simple averaging of translation and rotation to move the virtual windshield.

Another kind of collaborative manipulation consists in splitting the task among users [PBF02]. In this case, the number of DoF that each user can access and control is limited: one user controls rotation of the object while the other one is limited to translation. This can be compared to the Two-Hand "trackball" technique [CFH97].

Separate motions of several users' inputs (from several hands or users) can be used to define the final motion of a virtual object. However, due to the complexity of current VR interfaces, no universal collaborative solution has already been proposed to naturally apply a motion to a co-manipulated object. Therefore, in this chapter we will propose to extend these techniques by adding a third manipulation point for collaborative manipulation [ADL09].

5.2.3 Tangible devices

A tangible device (or prop) is a real object that users can hold to move a virtual object and feel a passive tactile feedback. Such tangible interfaces are often preferred by people over non-physical interfaces [SJF09]. However, several studies show that they do not always lead to better performance [HTP+97, WR99]. Nevertheless, passive tactile feedback can be used to increase presence and improve training effectiveness in virtual environments [IMWB01].

Many *adhoc* tangible interfaces have been proposed to mimic real objects in a virtual world such as in [HTP+97, SJF09]. In this case, users may have to hold a scale model for interaction with the virtual environment as in [HPGK94]. If some tangible interfaces let users use both hands, to our knowledge usual tangible interfaces are generally limited to single-user interactions.

Tangible User Interfaces (TUI) were designed to give a physical form to digital information [UI00]. They were shown to improve the manipulation of objects in many cases, in 3D or in 2D applications [IU97].

In virtual reality, a famous example is the neurosurgery visualization application of Hinckley et al. [HTP+97] in which a user holds two physical objects: a head doll in one hand and a plastic plane in the other hand. These tools are used to select and visualize more directly and more efficiently cuts in a 3D brain model. The Monkey [EPO95] is dedicated to the configuration of virtual human postures for the purpose of computer animation. Note that a TUI does not necessarily have to take the shape of the corresponding real object (iconic representation [UI00]) but it can also be an abstract representation of this real object (symbolic representation).

Tangible interfaces can also be designed for helping people to coordinate their movements during a collaborative manipulation. In [SJF09], Salzmann *et al.* propose a prop for two-user manipulation that maintains the users' hands at the same distance. As such, the prop acts as haptic link between them. As position and orientation are given by only one optical marker on the top of the prop, this technique is limited to one point of manipulation. The shape of the prop also restricts its use to cases where users use only one of their hands or keep their two hands very close.

We can further subdivide tangible user interfaces from the literature into two distinct types: non-reconfigurable TUIs and reconfigurable TUIs. Non-reconfigurable tangible user interfaces have a shape that cannot be modified, whereas reconfigurable TUIs are tangible interfaces whose shape can be modified. A first example of a reconfigurable TUI consists in assembling or removing physical blocks that progressively design and match 3D object shapes. Research on this topic was initiated with architectural applications [Fra95] and led to numerous TUIs such as the MERL bricks [AFM+99] or the ActiveCubes [WIA+04]. Another approach consists in deforming a malleable TUI. Users are then able to use pressure to build shapes in 3D modeling tools for instance [STP08]. Several reconfigurable TUIs follow an approach balancing between malleability and rigidity. The Senspectra TUI [LPI07] connects rigid balls by flexible joints to form an overall flexible structure. The Glume TUI [PLI06] connects malleable balls by a rigid structure. However, all these reconfigurable TUIs were built for specific applications, e.g. games or education, and never in the context of virtual reality and 3D virtual object manipulation. We assume that a good rigidity of a TUI could better mimic rigid "real" objects and provide a haptic passive link for collaborative manipulations between users. However, to the authors' best knowledge no previous work has been done in the area of reconfigurable tangible user interfaces to find a good balance between easy reconfigurability and rigidity for object manipulation in virtual environments.

5.3 3D virtual rays for collaborative manipulations

This section describes how we propose to help a user of a 3D collaborative virtual world to be aware of the constraints due to the other users during his collaborative interactions with the objects of the universe. As several users can interact simultaneously with the same shared interactive object, the behavior of such an interactive object can result from the addition of all the concurrent interactions, and it has to be explicitly shown to the users in order to let them understand why the co-manipulated object does not behave as it should (i.e. such as it would if there was only one user in interaction). So we present here some metaphors for collaborative interactions within 3D virtual worlds that can improve the awareness of such collaborative interactions for the users.

5.3.1 Introduction

Usual interaction metaphors such as virtual rays, virtual hands or 3D cursors are efficient for single-user interaction, but they must be adapted for multi-user simultaneous interaction on a single object. For example, when grabbing a virtual object with a virtual hand, the virtual object should follow the position of the hand until it is released. But if several users take control of the same object at the same time, with virtual hands, it is impossible to let the users move freely their virtual hand while making the grabbed object follow all these virtual hands. The only solution would be to use force-feedback input devices, in order to constrain the users to move their virtual hands in the same way. As force-feedback input devices are very expensive and do not usually offer a wide area for the movements of the users, we prefer to propose a solution to simultaneous interaction through new visual metaphors that can be driven with classical 3D input devices. These metaphors propose to give to the manipulated object a position which is the result of the action of all the users, and to give visual cues to the users to explain why the co-manipulated object is not where it would be if each user was interacting alone.

5.3.2 The targeted interaction tool: a virtual ray

We often use virtual rays for object manipulation. A virtual ray can be used to select and to interact with many kinds of virtual objects, for example thanks to the protocol presented in chapter 4.

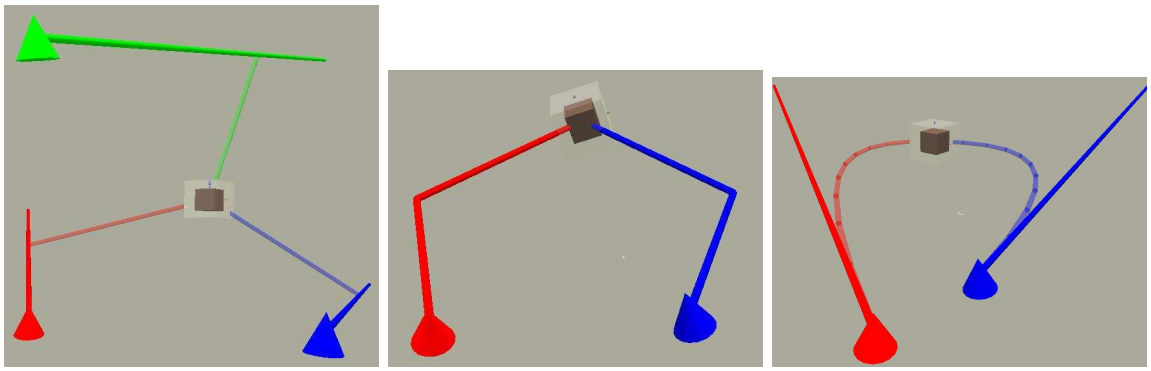
5.3.2.1 The ray with a rubber-band

Our first proposition to make a user aware of the difference between the “should be” position and the effective position of a virtual object manipulated through a virtual ray is to use a rubber-band. This rubber-band will make a link between the “should be” position and the effective position (see Figure 5.1(a)) of the manipulated object. As a result, each user is aware that there are some constraints on the manipulated object (here the constraints are due to other users who are currently co-manipulating the object), which explains why the co-manipulated object is not at the “should-be” position on his virtual ray. This solution can also be used for other interaction tools such as virtual hands or 3D cursors.

5.3.2.2 The creased ray

Our second proposition fits only virtual rays. Instead of using a rubber-band, we propose to resize the virtual ray and break it into two parts, in order to make the end of the ray reach the manipulated object, as illustrated in Figure 5.1(b).

This solution does not show to a user the “should-be” position of the co-manipulated object. There could be some ways to show the “should-be” position, for example with a half-transparent ghost of the object for each interaction tool currently co-manipulating it. Such a solution could be effective for every co-manipulation metaphor.



(a) (b) (c)

Figure 5.1: (a) Three virtual rays are interacting simultaneously with a virtual object — (b) The extremity of each creased ray reaches the manipulated object — (c) The bent rays make a link with the co-manipulated object.

5.3.2.3 The ray with a bent ray

Our third proposition is an evolution of the creased ray. Here we propose to bend the ray in order to have a progressive deformation of the virtual ray, instead of having only two segments. We also let the initial virtual ray always visible, for a better visual feedback, as illustrated in Figure 5.1(c). This additional bent ray is very similar to the one that has been proposed in 2006 by Riege et al. [RHWF06].

Here again, the user does not perceive where the object would be if he was the only user interacting with this object. It is easy to show this "should-be" position, either by changing the length of the straight virtual ray so that its extremity would show this position, or simply by using a rubber-band as proposed in section 5.3.2.1. In both cases this position is the position of the interactive object relative to the ray when the ray selects it.

5.3.3 Conclusion

These three metaphors have been first implemented in the GASP framework during a master thesis [DF02], with the very first version of our dialog protocol [DLT04]. They can be useful to make the users aware of the constraints that can be applied to interactive objects during an interaction. These constraints can occur because of collaborative interactions such as co-manipulation, or also because of mechanical constraints applied to the objects.

5.4 An asymmetric 2D Pointer / 3D Ray for 3D interaction within CVE

5.4.1 Introduction

What is the best technical solution for easy and natural 3D interaction within Virtual Environments (VE)? Most people will answer that it is immersion, but to obtain high quality immersion you need stereovision for the visualization, linked to a 3D tracking device in order to track the position of tools of the user and of his head. Indeed, such technical solutions allow the images to be generated such that virtual tools can be colocated with parts of the user's body or with the real tools he is using, so a user feels like his arms, hands, or tools were really embedded within the virtual environments. Furthermore, interaction metaphors that are usually used in this context, such as

virtual hands [PBWI96], virtual rays [BH97] or virtual 3D cursors [ZBM94], are interesting for Collaborative Virtual Environments (CVE) because they provide a natural 3D representation that is perceptible for the other users of the CVE. Due to this 3D visualization of the interaction tools, a user can be easily aware of the activity of the other users of the CVE. Nevertheless, 2D metaphors and input devices have also to be considered for 3D interactions because they are sometimes easier to use than 3D metaphors, as stated in [BCF+08].

However, a good immersion cannot be obtained without expensive hardware such as high-frequency video-projectors (for active stereovision) or double projectors (for passive polarized stereovision). Providing only stereovision is not enough to obtain a good immersion, because it cannot ensure a good co-location between the virtual tools driven by the users and the physical objects or body parts that the user uses to control the virtual tools. We need wireless tracking systems (optical, ultrasonic or magnetic) for head tracking, tools tracking and body parts tracking.

Without co-location, we consider that it would be difficult for somebody to use efficiently the classical 3D interaction metaphors, and that these metaphors will not be user-friendly. So perhaps basic 2D interaction tools such as a 2D pointer driven with a classical 2D mouse could be as efficient as the usual 3D metaphors for simple tasks such as object selection and manipulation (3D positioning, for instance).

Two problems arise when using such basic 2D interaction metaphors. First, when several users share a CVE, it will be difficult to make a user aware of the interactions of other users, because their 2D interaction tools will not be associated with any 3D virtual objects. Second, using a classical mouse will not fit semi-immersive environments when a user stands in front of a big image produced by a videoprojector, generally without any keyboard or 2D mouse.

This is the reason why we propose a new 2D pointer that will be associated with a 3D geometry in order to appear visually within the Collaborative Virtual Environment. This 2D pointer will be easy to use and will be driven by any device that can control a 2D position: for example a classical 2D mouse, a gamepad or a Nintendo wiimote remote gaming controller. The 3D geometry of this pointer will be a virtual ray, so other users can be easily made aware of the movement of this 3D ray, in the same way they can be made aware of the evolution of classical 3D interaction metaphors. This 2D Pointer / 3D Ray will use the classical ray-casting technique for object selection and manipulation. In this way, its behavior is similar to the aperture based selection technique [FHZ96] and to the technique developped in [WL97].

In order to show that our 2D Pointer / 3D Ray can be useful for selection and basic interaction tasks, we have made some experiments comparing four interaction techniques, which results are discussed in [DF09].

5.4.2 The asymmetric 2D Pointer / 3D Ray

Our idea is to use a 3D virtual ray that would be as easier to drive than the classical 2D mouse pointer. The result looks like a classical 2D pointer moving on the surface of the screen. In fact it is a quite thin and long 3D virtual ray, moving near the viewpoint of the user, staying always at the same depth, which orientation is calculated in a way that its projection on the screen is always a small spot.

As shown in Figure 5.2, the 2D device used to control the pointer will provide the Xc and the Yc values, and the Zc value is a chosen one, so the rho and $theta$ values can be calculated this way, if the rho angle (the heading) is first applied around the Y axis and then the $theta$ angle (the elevation) is applied around the X' axis :

- $rho = atan(-Xc/Zc)$
- $theta = atan(Yc/sqrt(Xc * Xc + Zc * Zc))$

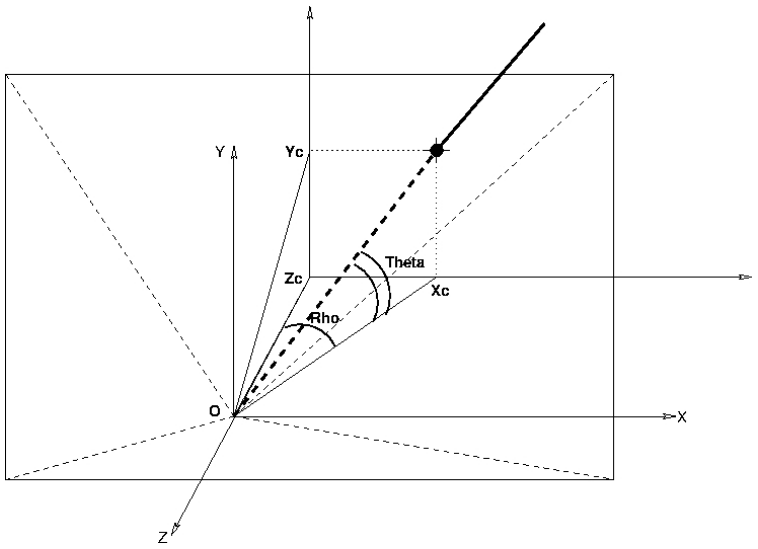
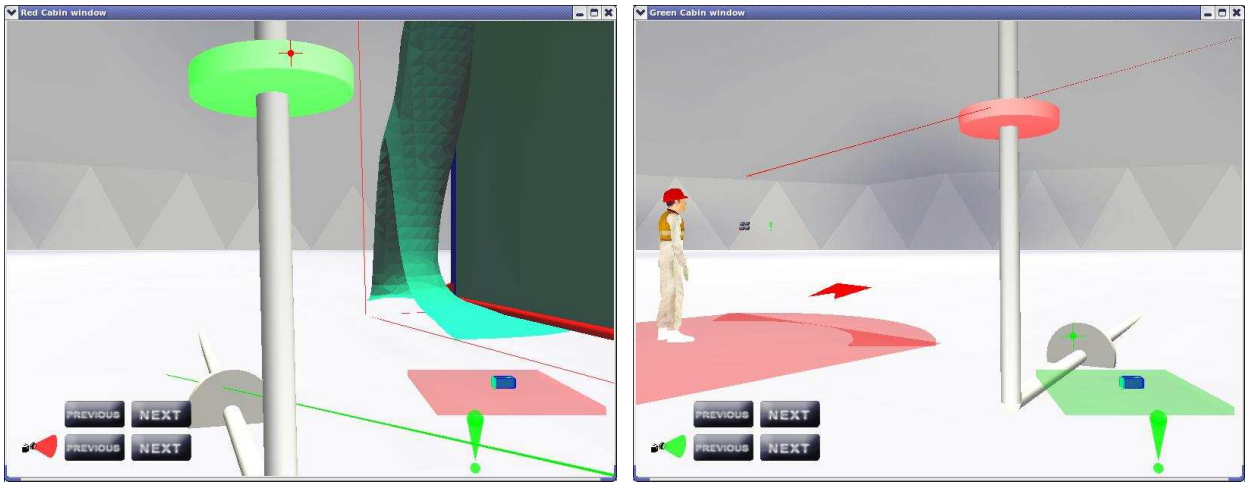


Figure 5.2: Projection of the 3D Ray as a small spot on the screen.

This way, the user of the 2D Pointer / 3D Ray will always feel that he is using a 2D pointer (see Figure 5.3(a)), while other users will see a 3D virtual ray moving thanks to the action of the first user (see Figure 5.3(b)). So it is quite easy to use for the first user, and quite easy to understand by the other users.



(a) (b)

Figure 5.3: (a) On the left, user 1 moves a 3D slider with his red 2D pointer and he sees the green 3D virtual ray of user 2 ready to select another slider. (b) On the right, user 2 is ready to select a slider with his green 2D pointer while he is looking at user 1 moving a slider with his red 3D virtual ray.

This 2D Pointer / 3D Ray is completely independent from the hardware device that will be used to drive it: either a classical 2D mouse, or a game pad, or any device able to provide 2D coordinates, or even a graphical 2D user interface.

As the 2D Pointer / 3D Ray is turning around the closest extremity of the virtual ray, the movements of a manipulated object can also be affected by a small rotation and will not stay at the same Z coordinate within the user's coordinate system, except if we force it to preserve its relative

orientation and Z coordinate.

This metaphor can be easily extended to 3D movements within the user's coordinate system: the X and the Y coordinates are directly provided by the physical device used to drive the 2D pointer, and the Z coordinate can be changed by moving the manipulated object along the 3D ray. To achieve such a translation along the virtual ray, the device used to drive the 2D pointer must also provide the information needed to calculate the Z coordinate, or it can be associated to another device providing this value. For example, this Z coordinate can be obtained thanks to the wheel of a 2D mouse, or some buttons of a gamepad.

A rotation of the manipulated object within the user's coordinate system can also be calculated with additional devices, for example the keyboard or some buttons or joysticks of a gamepad.

We consider our technique as an egocentric interaction metaphor using a pointer as described in [PWBI98]. As our 2D Pointer / 3D Ray is a tool associated to the user's viewpoint, the user carries this interaction tool with him when he navigates within the VE, in the same manner as 3DM [BDHO92]. So as the 2D Pointer / 3D Ray moves with the viewpoint when the user navigates, the object that has been grabbed by the moving tool navigates also within the VE, which is another complementary way to provide a new position and orientation to this manipulated object.

Last, the 2D Pointer / 3D Ray can simply be used as a classical 2D pointer to trigger somme elements of a 3D GUI that could be carried by the user, in order to control the state of the application.

So, according to Hand [Han97] who separates virtual interactions into 3 categories: 3D interaction (selection and manipulation), navigation and application control; we see that our 2D Pointer / 3D Ray, carried by the user, is well suited for these three kinds of interactions.

5.4.3 Conclusion

We have proposed a new metaphor for 3D interaction within Collaborative Virtual Environments: the 2D Pointer / 3D Ray, which associates a 3D representation with a 2D pointing device (for example a 2D mouse). This metaphor allows an asymmetric collaboration between users immersed within a CVE (via stereovision and head-tracking) and users simply sitting in front of the screen of their workstation. The user without immersion will interact as easily as if he had a simple 2D pointer, as the associated 3D ray (a 3D virtual ray) will be continuously moved and oriented in a way that its projection on the screen of the user will always be a small spot. The other users of the CVE will be made aware of the action of this user thanks to the movements of his associated 3D virtual.

5.5 The SkeweR

In this section we introduce an interaction technique for two users to move the same virtual object simultaneously in a VE. The technique is called the "SkeweR" technique [DLT06] since each user manipulates the object by one crushing point, like handling one extremity of a skewer. When using the SkeweR technique, the final motion of the manipulated virtual object is based on a combination of the two translation motions of the users.

5.5.1 Introduction

The SkeweR technique allows multiple users to select and manipulate virtual objects simultaneously in a VE. Most of the previous interaction techniques were designed for the multiple users to manipulate the virtual object by translating or rotating its geometrical center (or gravity center). On the contrary, we propose to take into account the size and geometry of the object, in order to obtain a more natural interaction. Indeed, we propose to move the virtual object by grabbing

any part of it, at any location. The SkeweR technique can be first considered as an equivalent (in the field of 2-user collaborative manipulation) of the "grab-an-carry" tool described by Cutler et al. [CFH97] which was designed for 2-hand manipulation. However, the principle of the SkeweR technique can also be extended, in order to be used by 3 (or more) users. The SkeweR technique can also be compared with the techniques developed for shared haptic interaction [BHSS00]. However, the SkeweR technique does not use force-feedback and physical models, but simply uses kinematics and information of motions of the users. The SkeweR technique can tolerate large offsets between the position of the user's hand and the position of the shared object in the simulation.

5.5.2 Concept

The SkeweR technique uses only the translation motion of the users. These motions are measured by position sensors. They are used to move one virtual 3D cursor per user in the VE. These 3D cursors are used to activate some "crushing points" on the manipulated object (see Figure 5.4). We will assume that a virtual 3D cursor stays at the same position than its associated crushing point.

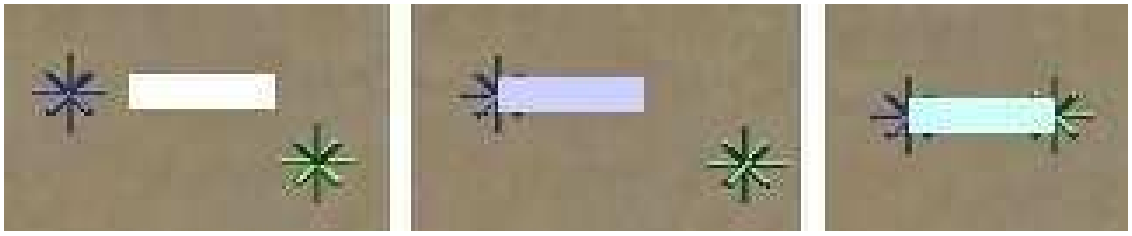


Figure 5.4: Two users reach and crush a parallelepiped object with their virtual 3D cursors

When associated with the crushing points, the 3D cursors of the two users become the extremities of a virtual skewer which is used to hold and move the interactive object. As displayed on Figure 5.5, a virtual object (parallelepiped) can be turned and rotated easily in a VE without using any rotation motion from the users.

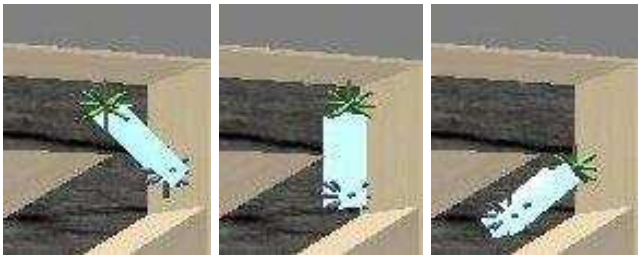


Figure 5.5: 2-user collaborative manipulation of an object with the SkeweR technique in one turn of a virtual maze

The SkeweR technique uses only the translation motions of the two users to move the objects within a complex 6 DOF movement. The scheme is thus different from the techniques described in section 5.2, making the SkeweR technique a hybrid technique.

5.5.3 Using only one crushing point

It is possible to change the orientation of the manipulated object with only one crushing point: once a crushing point (associated to a 3D cursor) is selected and activated on the manipulated

object, the translation motions of the user are used to move the cursor and then to compute a new rotation increment for the object. At each time step, we consider an elementary straight motion of the cursor, and we compute an elementary rotation to apply to the object. If Po is the geometrical center of the object, Pc_t the position of the 3D cursor at time t, Pc_{t+1} the position of the 3D cursor at time $t+1$, then the elementary rotation to apply to the interactive object will be a rotation of a θ angle around the ω axis, i.e. the axis which is passing by Po and orthogonal to the plane defined by Po, Pc_t and Pc_{t+1}. The θ angle and ω axis are computed as follows:

$$\theta = \arccos\left(\frac{\overrightarrow{PoPc_t} \cdot \overrightarrow{PoPc_{t+1}}}{||\overrightarrow{PoPc_t}|| \times ||\overrightarrow{PoPc_{t+1}}||}\right) \quad (5.1) \qquad \overrightarrow{\omega} = \overrightarrow{PoPc_t} \wedge \overrightarrow{PoPc_{t+1}} \quad (5.2)$$

The change in position of the manipulated object is achieved by constraining the position of the crushing point (relative to the object) to follow the position of the cursor. Let us assume that R is the rotation used to determine the absolute orientation of the manipulated object. R is obtained by the accumulation of the elementary rotations defined previously with Equation 5.1 and Equation 5.2. The new position of the object is then obtained using Equation 5.3.

$$Po = Pc - R \cdot Pc/Po \quad (5.3)$$

In this equation, (Pc/Po) represents the local position of the 3D cursor relatively to the center (Po) of the manipulated object, in the relative frame of this object. This position is the same than the position of its associated crushing point, and it remains constant as long as the user does not choose to manipulate the object thanks to another crushing point. As a result, this technique gives the user the impression that he is pulling the object with a virtual cord.

5.5.4 Extension to 2 crushing points

With two crushing points, each user may control the motion of the manipulated object as if he was holding one extremity of a virtual skewer. The final position of the interactive object is computed using Equation 5.4, in which $Pc1$ and $Pc2$ are the absolute positions of the cursors of the 2 users, and $(Pc1/Po)$ and $(Pc2/Po)$ are the local positions of these 3D cursor relatively to the center of the manipulated object, in the relative frame of this object. The final position reached by the object corresponds to the middle of the two positions obtained when using the two crushing points.

$$Po = \frac{Pc1 + Pc2}{2} - R \cdot \frac{Pc1/Po + Pc2/Po}{2} \quad (5.4)$$

Here again, the expression $((Pc1/Po + Pc2/Po)/2)$ remains constant as long as one user does not choose to manipulate the object thanks to another crushing point. Since the four points $Pc1_t$, $Pc1_{t+1}$, $Pc2_t$ and $Pc2_{t+1}$ do not always remain in the same plane, the final rotation applied to the interactive object now uses the vector v defined by Equation 5.5. This vector is defined using the absolute position of the two cursors. The new θ rotation angle and the new ω rotation axis are then given respectively by Equation 5.6 and Equation 5.7.

$$\overrightarrow{v} = \overrightarrow{Pc1Pc2} \quad (5.5) \qquad \theta = \arccos\left(\frac{\overrightarrow{v_t} \cdot \overrightarrow{v_{t+1}}}{||\overrightarrow{v_t}|| \times ||\overrightarrow{v_{t+1}}||}\right) \quad (5.6) \qquad \overrightarrow{\omega} = \overrightarrow{v_t} \wedge \overrightarrow{v_{t+1}} \quad (5.7)$$

5.5.5 With only 2 crushing points: one DOF is missing...

It is not possible for the two users to rotate the virtual object around the axis of the skewer, since the rotation axis remains always orthogonal to the skewer's axis. To do so, the users have to stop their current motions and change the positions of their crushing points, which means change the axis of the skewer. We can imagine complementary techniques to make up for the lack of this sixth degree of freedom, and apply rotations around the skewer axis. First, we could use the rotation motions of the users around the skewer's axis straightforwardly, as measured by orientation trackers (similar to the "grab-and-twirl" tool referenced in [CFH97]). However, this solution can not be implemented with 3 DOF trackers. Second, when pressing a button, users could be enabled to move their 3D trackers along the skewer's axis to define the sign and angle of a rotation as a function of the translation motion of each user along the axis. For example when the 3D trackers would get closer the object would rotate positively around this axis, and it would rotate negatively otherwise.

5.5.6 Extension to 3 crushing points, or more...

The proposed technique could be extended to the use of more crushing points. This could indeed allow a better control of the motion of the interactive object. To control the orientation of an object, the best solution is to use 3 crushing points (not aligned), as these 3 points can easily be used to define exactly the orientation of the manipulated object. It will be detailed in section 5.6. Indeed, these 3 crushing points define a plane embedded in the manipulated object and we can easily compute one vector orthogonal to this plane. Each time one crushing point (i.e. one 3D cursor) is moved, it is possible to compute a new orthogonal vector and to determine the exact rotation that transforms the initial vector into this new one. Furthermore, the position of this object could be computed as the average of the positions of the 3 crushing points. In the case of a manipulation of three crushing points by one user, one or two crushing points could be used to fix temporarily some position constraints, and the remaining crushing points would be used to move the object and change its orientation. With four crushing points or more, the orientation of the interactive object might become over-constrained. Thus, with more than three crushing points, one must find arbitrary simplifications to apply changes in orientation to the object. Future work must be done to manage the computation of the orientation in this over-constrained case.

5.5.7 Preliminary experimental setup

This technique has been implemented thanks to our OpenMASK platform within our Reality Center, with magnetic trackers. We have made some preliminary tests, asking two users to move a parallelepiped object inside a 3D maze. Our first observations indicate that the SkeweR technique seems very intuitive and natural to use. Further experiments would be necessary to measure if this technique is easy and efficient enough for collaborative manipulations of 3D virtual objects.

5.5.8 Conclusion and perspectives

The SkeweR technique is a 3D interaction technique for 2-user collaborative manipulation of virtual objects. This technique enables two users to move simultaneously the same virtual object in a 3D environment. The technique uses and combines the translation motions of the two users, as measured by 3 DOF position trackers. Each user manipulates the object by one "crushing point", like handling the extremity of a skewer. This technique can also be used with only one crushing point, and could also be used with 3 or more crushing points.

5.6 The 3-hand manipulation technique

5.6.1 Concept

We propose a new 3D interaction technique for 6 DOF multi-user collaborative manipulation of 3D objects. Our technique enables the determination of virtual object position and orientation through only positions of three non-aligned manipulation points on this object. These manipulation points can be used naturally, in a realistic way, by three different "hands" of two or three users.

5.6.2 Manipulation and visual feedback

Hands are represented by pointers. When a hand is close enough to the object to manipulate, ray-casting from the hand gives an intersection point with the object. This point is called a *manipulation point*. If a user starts a manipulation, a virtual ball is added to display the location of the manipulation point. In addition, a rubber band is added between the virtual ball and the hand to avoid any ambiguity concerning its owner and to display the distance between the hand and the manipulation point (see Figure 5.7). A screenshot of the virtual environment with three hands manipulating a hood is given in Figure 5.6.

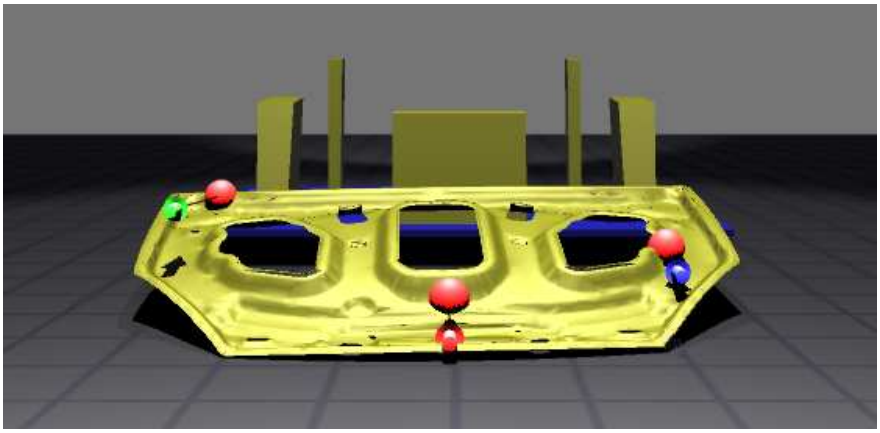


Figure 5.6: Screenshot of a 3-Hand Manipulation of a virtual hood.

The rubber band drawn between a hand and its manipulation point is elastic and its color varies according to the distance between the hand and the manipulation point. The rubber band uses a green-yellow-red code: the farther a hand is from its associated manipulation point, the more the rubber band becomes redder. With such a feedback, users' hands are expected to remain close to their manipulation point to avoid instabilities during the manipulation.

5.6.3 Computation of manipulated object's motion

The manipulated object motion can be computed in different ways using input motions of the three users' hands. One solution consists in making the manipulation points stay as close as possible to the hands. At the beginning of the manipulation, hands positions H_1, H_2, H_3 correspond to positions of their contact points P_1, P_2, P_3 with the manipulated object. These contact points are the manipulation points and are illustrated in Figure 5.8.

When users move their hands to H'_1, H'_2, H'_3, the $\vec{T}$ translation to apply to the initial position P_c of the manipulated object is computed as follows:

$$H_0 = \frac{H_1 + H_2 + H_3}{3} \quad ; \quad H'_0 = \frac{H'_1 + H'_2 + H'_3}{3} \quad ; \quad \vec{T} = \overrightarrow{H_0 H'_0}$$

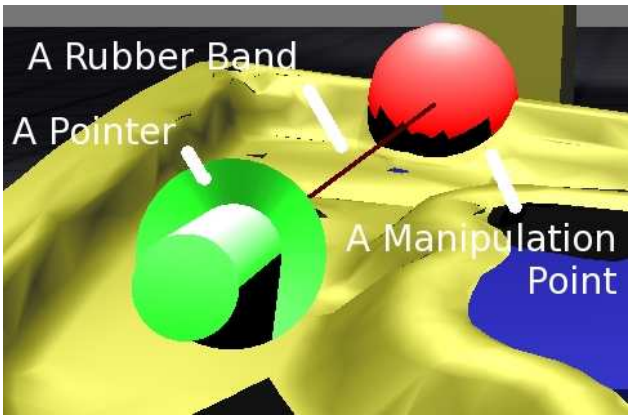


Figure 5.7: A rubber band between a pointer and a manipulation point.

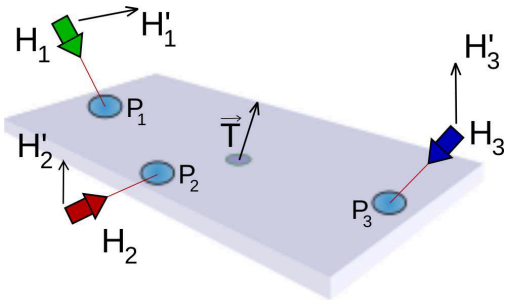


Figure 5.8: Computation of virtual object's motion ($\vec{T}$) given the three hand's motions.

To compute the rotation difference with the initial plane orientation, the first step consists in computing the rotation ($\vec{R_1}$, α_1) that transforms $\vec{i}$ into $\vec{i'}$ (see Figure 5.9):

$$\vec{i} = \overrightarrow{H_0H_2}\ ;\quad \vec{j} = \overrightarrow{H_0H_3}\ ;\quad \vec{i'} = \overrightarrow{H'_0H'_2}\ ;\quad \vec{j'} = \overrightarrow{H'_0H'_3}\ ;\quad \vec{R_1} = \vec{i} \times \vec{i'}\ ;\quad \alpha_1 = \arccos\left(\frac{\vec{i} \cdot \vec{i'}}{\|\vec{i}\| * \|\vec{i'}\|}\right)$$

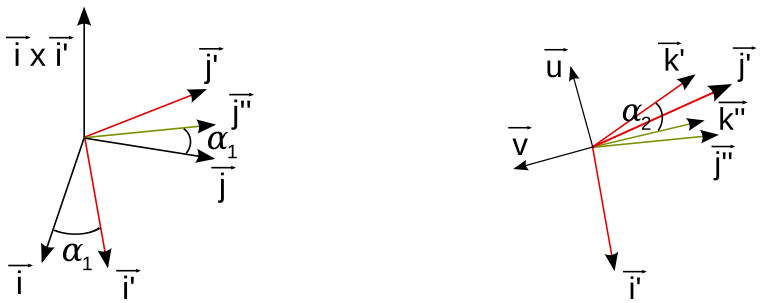


Figure 5.9: Rotation $(\vec{R_1}, \alpha_1)$ and rotation $(\vec{i'}, \alpha_2)$.

This rotation transforms $\vec{j}$ into an intermediate vector $\vec{j''}$: $\vec{j''} = (\vec{R_1}, \alpha_1) * \vec{j}$

The second step consists in computing the rotation ($\vec{i'}$, α_2) that transforms $\vec{j''}$ into $\vec{j'}$. After constructing a new 3D orthogonal basis ($\vec{u}$, $\vec{v}$, $\vec{i'}$), this angle is calculated as follows:

$$\vec{k''} = (\vec{j''} \cdot \vec{u}) * \vec{u} + (\vec{j''} \cdot \vec{v}) * \vec{v} \; ; \quad \vec{k'} = (\vec{j'} \cdot \vec{u}) * \vec{u} + (\vec{j'} \cdot \vec{v}) * \vec{v} \; ; \quad \alpha_2 = \arccos\left(\frac{\vec{k''} \cdot \vec{k'}}{\|\vec{k''}\| * \|\vec{k'}\|}\right)$$

The difference between the current and the initial orientation of the object is then obtained by combining these two rotations: $(\vec{i'}, \alpha_2) * (\vec{R_1}, \alpha_1)$.

If the first triangle, defined by the positions of the three hands, and the second triangle, defined by the three initial positions of the manipulation points, do not keep the same shape then the roll angle (rotation around the axis orthogonal to these triangles) is the best possible approximation, otherwise this roll angle value is exact.

Another solution for implementation is to use three "point-to-point" constraints of a physics engine like Bullet [Bul] or PhysX [Phy]. A constraint is dynamically added between a hand and a manipulation point. Here, if the triangles do not keep the same shape, there can be some small inconsistencies for the roll angle.

In both cases, the use of a colored rubber band can help users to keep their hands close to the manipulation points.

5.6.4 Implementation

The 3-Hand Manipulation technique was implemented in a virtual reality center involving ARTracking markers [ART]. The five ART infrared cameras placed around a large screen were tracking hand positions in 3D space. Users were located in front of this large screen.

Two or three people could manipulate simultaneously a virtual hood to place it on a support. This hood had holes that users had to align with the support. This task is inspired by an assembly task faced in automative industry.

Figure 5.10: Three users manipulating a virtual hood with the 3-hand manipulation technique.

Physics and collisions in the virtual environments were implemented using the Bullet physics engine. Hands were manipulating objects through Bullet constraints.

During the manipulation, we observed that people needed to communicate a lot: to start or end the action, and to decide where to move the hood. All users found the 3-Hand Manipulation technique natural. During a 2-user manipulation, one user had notably the impression that "he was moving his hands in the air as he would have done with a real object".

5.6.5 General conclusion and perspectives

We have presented a new 3D interaction technique for multi-user collaborative manipulation of virtual objects called "3-Hand Manipulation" [ADL09]. This technique relies on the use of three manipulation points that can be used simultaneously by three different "hands" of two or three users. The three translation motions of the manipulation points fully determine the resulting 6 DOF motion of the manipulated object.

The considered task consisted in assembling a virtual hood on a support. Different configurations were tested with two or three users. First user feedback suggested that the technique was suitable for collaborative manipulation, which was validated through the distant collaborative manipulation of a clipping plane between 2 users for analysis of scientific data [FDGS12].

5.7 A Reconfigurable Tangible Device for 3D object manipulation

Although most collaborative systems support simultaneous manipulation of different objects by different users, generally only one user at a time can manipulate a virtual object. Interaction metaphors that are usually used for single-user 3D interaction, such as virtual hands, virtual rays or virtual 3D cursors, must be adapted for collaborative 3D virtual manipulation. This is why we have proposed a new physical device named Reconfigurable Tangible Device (RTD) [ADL10b] to enable collaboration on a shared virtual object through a tangible device and precise positioning of users' real hands. The RTD maintains the distance between users hands. Besides users can modify the shape of the RTD to better fit the virtual object they intend to manipulate.

5.7.1 Concept

Our main objective is to match numerous shapes of 3D objects, in various contexts of manipulation in virtual reality. We consider that such a Tangible User Interface (TUI) should be reconfigurable simply and quickly, to match many kinds of virtual objects, and to avoid interrupting the users' activity when being reconfigured. To enable single but also multiple-user manipulation, we believe that such a TUI should provide rigidity and act as haptic passive link between multiple hands.

For this purpose, we have designed a novel concept of TUI called the Reconfigurable Tangible Device (RTD) that is both rigid and reconfigurable. It proposes a generic and universal physical user interface made up of points rigidly linked together. The manipulation points are *handles* that form a simple shape that can be modified. The shape corresponding to the points roughly sketches the shape (or mesh) of the virtual object that is manipulated by the user(s). The RTD provides rigid, but easy-to-stretch, physical links between each manipulation point (or physical handles). The RTD has been designed to support two (or more) users in interacting together with the same virtual object. In this case, rigid links between users' hands act as a passive haptic feedback, which could improve the collaborative manipulation [SJF09].

5.7.2 RTD-3: Reconfigurable Tangible Device with 3 points of manipulation

The first instance of RTD that we have developed is a triangular version called RTD-3 (see Figure 5.11). This reconfigurable triangle is made up of three arms connected together by a pivot. As a result, it supports three points of manipulation (handles). It can be considered as a physical and reconfigurable version of the 3-Hand Manipulation technique introduced by Aguerreche et al. [ADL09]. This configuration fully determines the position and orientation of any attached virtual object through the positions of three non-aligned manipulation points placed on it. More generally, the RTD-3 could enable to grab any part of an object to manipulate even if it is in an inner or outer part of the virtual object. As a result, two or three users can move, resize or reshape a virtual object thanks to this device.

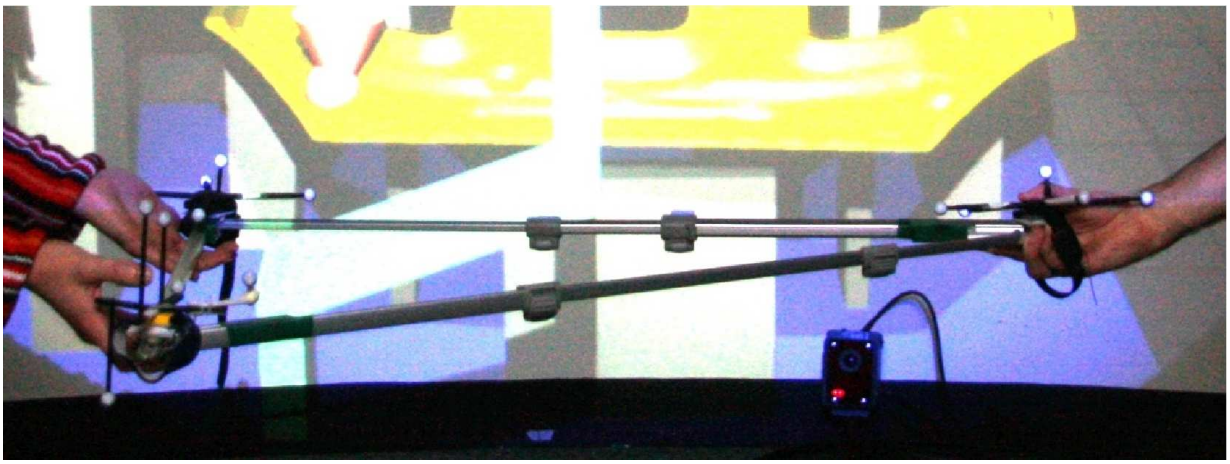


Figure 5.11: Reconfigurable Tangible Device with 3 points of manipulation (RTD-3): A reconfigurable triangle.

The RTD-3 is made up of three branches connected together by a pivot. Each arm of the RTD-3 can be compressed or stretched (see Figure 5.12) by pulling a button to unlock/lock an arm. When users want to attach the device to an object, they start to set the relevant branch lengths. Then they move the three virtual points associated with the device in order to touch the virtual object to manipulate. After selecting the object, users can begin to manipulate it.

Varying the lengths of the arms and using many angles let users obtain small or large triangles with lengths from 38 cm up to 95 cm, and angles varying from 20 degrees up to 130 degrees. As a result, the RTD-3 can be used to match various basic shapes of virtual object (see Figure 5.12). Users are able to grasp flat objects but also long, round or cubic objects. Virtual objects can be grasped horizontally or vertically.

5.7.3 RTD-4: Reconfigurable Tangible Device with 4 points of manipulation

The second instance of RTD that we have developed uses four points of manipulation (see Figures 5.13 and 5.14). This instance is called RTD-4, it enables users to compose various and complex shapes of 3D objects in 2D or 3D, i.e., quadrilateral or tetrahedral (see Figure 5.13). It is made of four stretchable and rigid arms (same type as the RTD-3) connected together by articulated joints that enable the overall structure to become non-planar. This feature is extremely useful with articulated objects, such as a door to open for instance. An example of how to use the RTD-4 is given in Figure 5.14 in which the RTD-4 approximately matches the shape of a virtual chair.

5.7.4 Implementation details

The two versions of the RTD introduced in this paper have been tested within a virtual reality center involving an ART [ART] optical tracking system. Virtual environment was displayed on a large stereoscopic screen. An optical marker was placed on each handle (manipulation point) of the RTD. Infrared cameras were placed around users to track positions and orientations of the optical markers. Each marker was associated with a virtual pointer in the virtual environment.

We built both RTDs with low-cost camera tripods that we have disassembled. For RTD-3, the three telescopic arms of a tripod are linked together by hinges. For RTD-4, four tripods have been used: 4 arms and 4 tripod camera supports (or tripod 'heads'). Each tripod camera support provides 2 hinges which leads to RTD with many articulations that can be turned into 3D. The resulting RTDs are both rigid and light: 200 g for RTD-3 and 400 g for RTD-4.

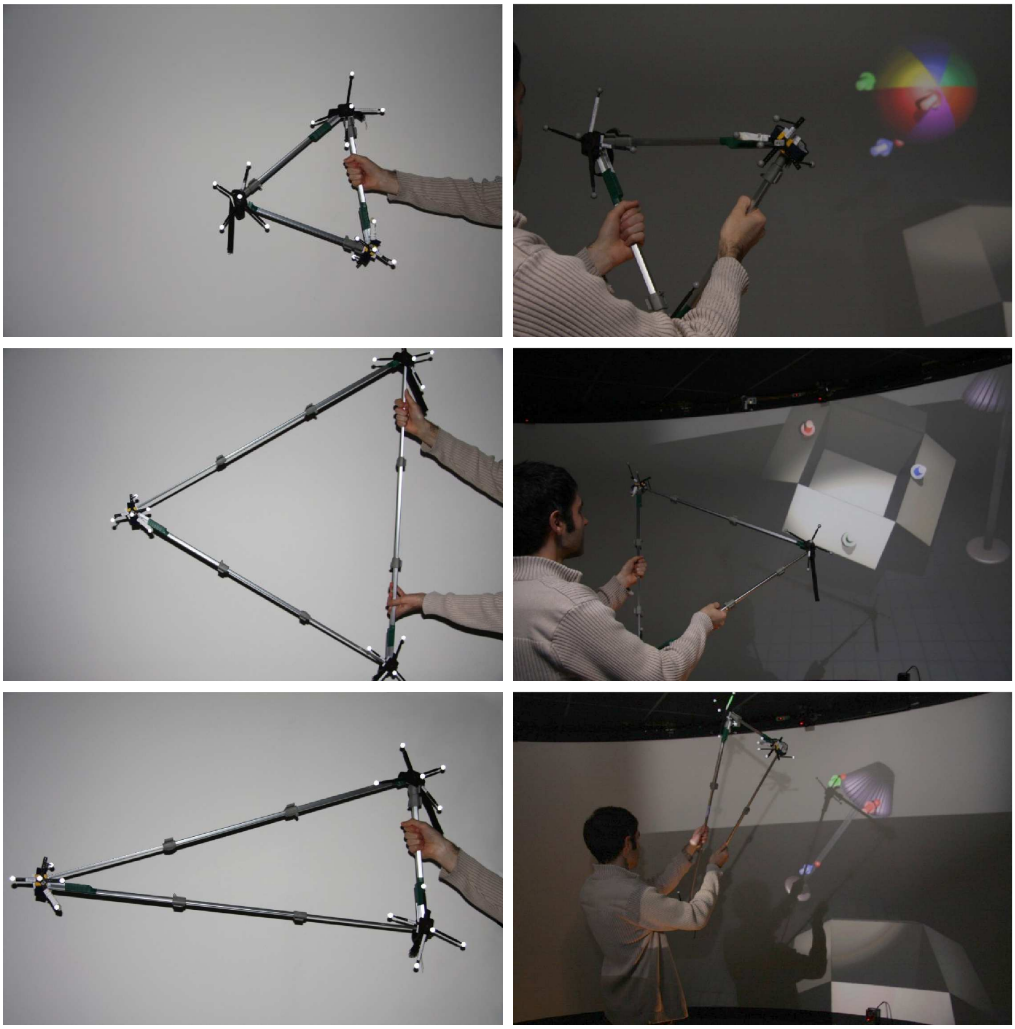


Figure 5.12: Different configurations of the Reconfigurable Tangible Device (RTD-3) (on the left) to match the manipulation of the corresponding objects (on the right).

In collaborative two-user situations with the RTD-3, users can move the triangle seamlessly together by applying movements to the device. The two users can naturally and easily use their two hands to hold the RTD-4 by all its corners.

When users want to attach the RTD to a virtual object they first set the relevant RTD shape, i.e. adjust the various lengths of the arms. They can then move the virtual pointers associated with the manipulation point of the device in order to touch the desired virtual object. After selecting the object (button click), users can begin to manipulate it.

5.7.5 Manipulation examples

In virtual reality, we can give different examples of manipulation in which our concept of Reconfigurable Tangible Device could be tested:

- **Classical manipulation of 3D objects**. The RTD is used to change the global position and orientation of the virtual object (classical case described above). Users can hook the RTD on any part of the object, provided that the RTD can be deformed to match the shape of this object. Movements of the RTD are directly transferred to the virtual object.

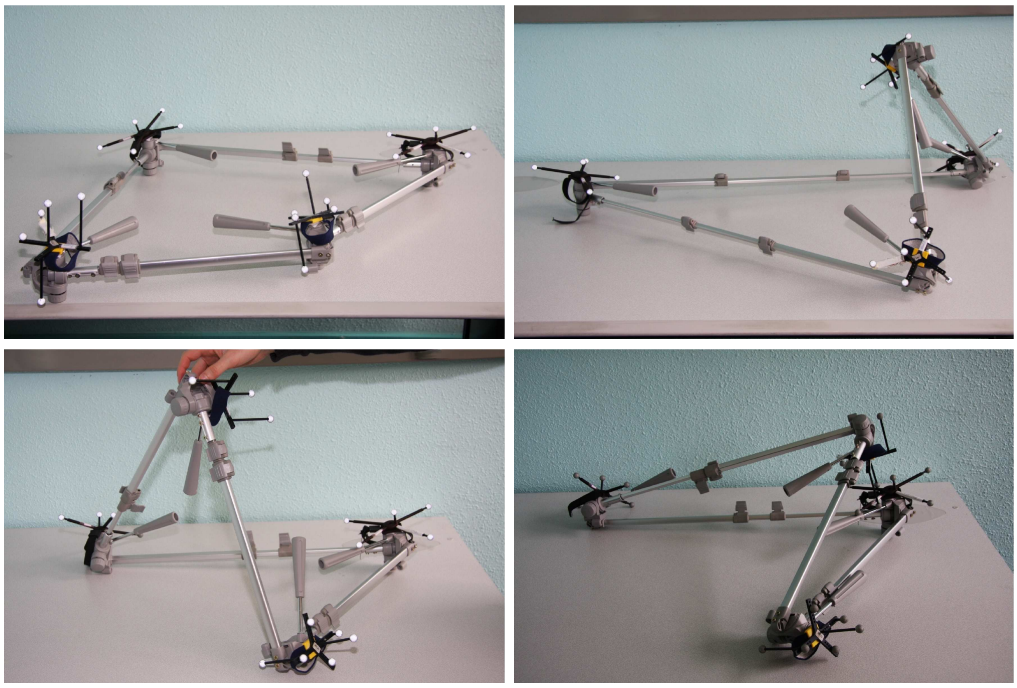


Figure 5.13: Reconfigurable Tangible Device with 4 points of manipulation (RTD-4): Different configurations and different shapes.

- **Manipulation of articulated objects**. The RTD is used to act on one part of the virtual object that is linked with the other parts through an articulated joint. Movements of the RTD are transferred to the manipulated part of the object, and movements of other parts of the object are generated through the articulated joint.
- **Deformation of objects**. The RTD is used to modify and deform the shape of a virtual object (e.g., vertices of the 3D mesh). In this case, changing the shape of the RTD could modify the shape of the manipulated object using control points or direct manipulation of vertices of the mesh.

5.7.6 Evaluation

The objective of our evaluation was to compare the RTD-3 with two classical techniques for collaborative virtual manipulation: the Mean and Separation of DoF techniques. The proposed task is a "pick-and-place" task involving two users in the manipulation of a virtual car hood described in [ADL10a] and illustrated in Figure 5.15. We collected task completion time, number of collisions, distance covered by the virtual hood and by the users' hands, and a questionnaire on users' subjective preferences. Preliminary results of this evaluation about subjective preferences have been briefly presented in [ADL10b], and the full details have been presented in [ADL11].

5.7.7 Conclusion

We have introduced the novel concept of a Reconfigurable Tangible Device for 3D object manipulation in virtual environments. It is a physical interface that can be altered by the user to appropriately match the shape of a virtual object. It is made up of handles (or manipulation points) rigidly linked together by arms. This device can be reconfigured at any time as its arms can be compressed or stretched by users at will. The RTD can be used by multiple users, who can grasp different parts of the interface. We have developed two versions of the RTD with three (RTD-3)

Figure 5.14: Example of configuration of the RTD-4 for manipulating a virtual chair.

or four (RTD-4) manipulation points. Both have been implemented and tested within a virtual reality center for collaborative manipulation of 3D objects.

We conducted an experiment to compare the RTD-3 with classical collaborative interaction techniques: the Mean of interactions and the DoF Separation. Objective results show that the RTD-3 ends with slower task completion time than the Mean technique probably due to the complex movements of the two-handed user, and that the RTD-3 is as precise as the Mean technique. Subjective results show that our technique provides a better sense of immersion and better realism. According to users, the RTD-3 provides a better knowledge transfer to the real world.

The evaluation suggests that the RTD-3 could be used in many collaborative applications such as for training people in virtual environments to do collaborative assembly or maintenance tasks.

Future Work concerns first more evaluations of our techniques, notably to compare the two versions of our reconfigurable tangible device (RTD-3 vs. RTD-4). We would also like to study the use of the RTD in different scenarios and use-cases such as object deformation by means of control points associated with the manipulation points of the RTD.

We could also construct other instances of RTD with more points, for example to allow more than two users to interact with the same virtual object. We also plan to use the RTD for dynamically resizing or reshaping virtual objects.

5.8 Conclusion and future work

In this chapter we have presented our contribution to the construction of new metaphors for interaction within Collaborative Virtual Environments (CVE). They are the evolution of usual 3D interaction metaphors that have been adapted to fit with collaborative interactions, and to make users aware of this collaboration. All these metaphors have been constructed using the protocol of dialog between interaction tools and interactive objects that has been presented in chapter 4.

We plan to go on proposing new 3D collaborative interaction metaphors in order to improve collaborative experience in CVE, which is still a very relevant topic as it was the core subject of the 3DUI 2012 contest. We participated to this contest by proposing some solutions based on the Collaviz framework in order to enhance collaboration between two users [NFD12] and we are now evaluating these solutions and proposing new ones dedicated to fully immersive collaboration.

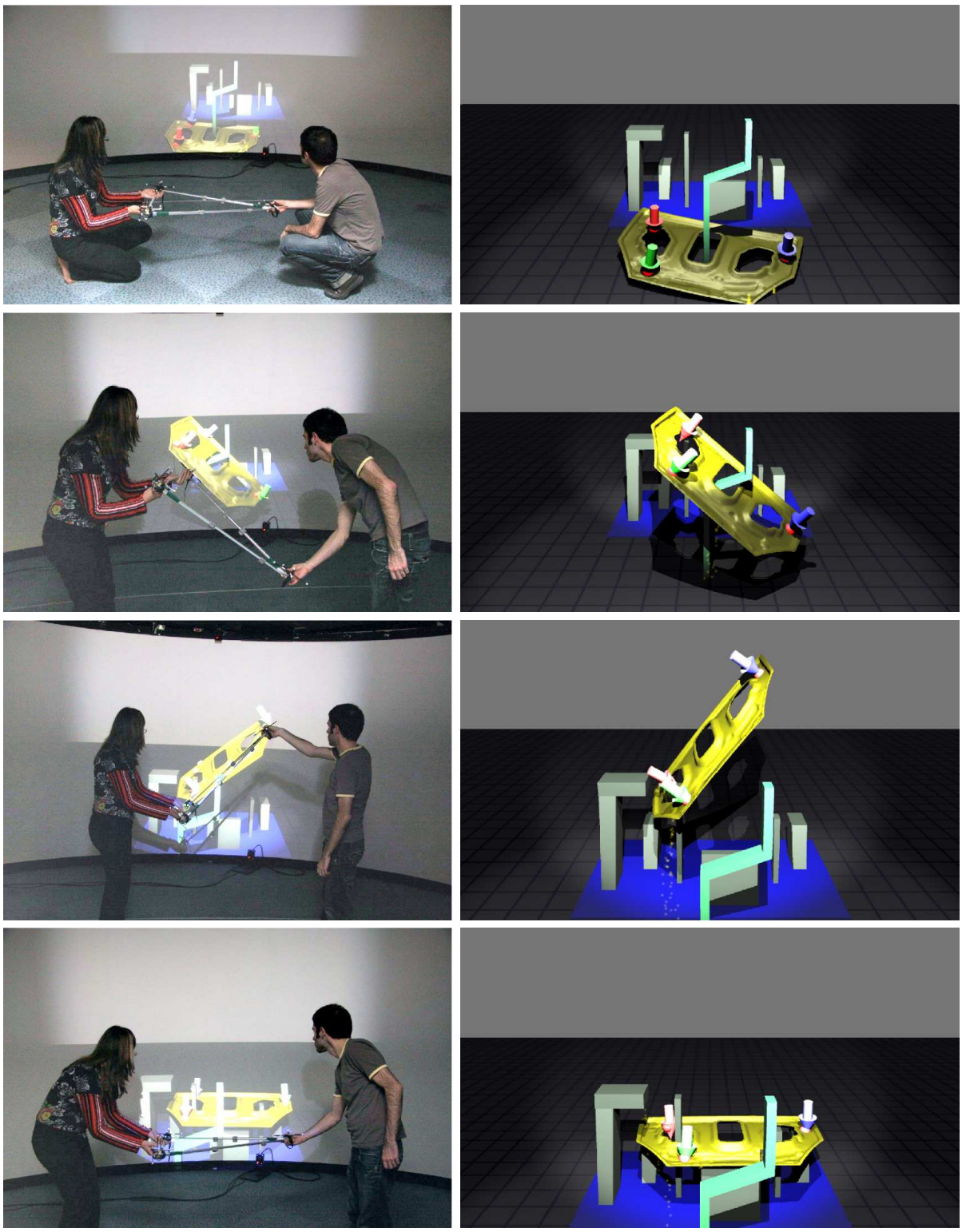


Figure 5.15: Experimental task. Column on the left: two users are achieving the task with the RTD-3. Column on the right: movements of the virtual car hood. Steps are as follows: 1) initial position, 2) passing the "elbow" of the "Z-shape", 3) passing between the "T-shape" and a stem, 4) reaching the final position.

Chapter 6

Modeling Users' Physical Workspaces

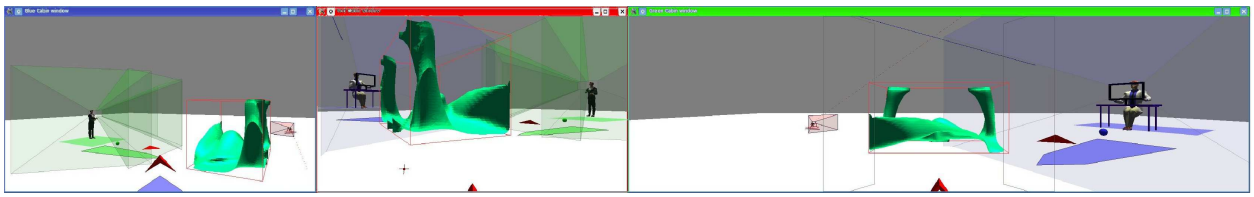


Figure 6.1: Three users with their own IIVC navigating and interacting within a Multi-Scale Collaborative Virtual Environment: each user can perceive other users' physical workspaces

6.1 Introduction

Nowadays Virtual Reality (VR) applications are used in many different fields such as industrial design, scientific data visualization and training, etc. Each kind of application requires specific interaction techniques and can be used on various physical environments from full immersive devices to "non-immersive" devices. To improve user presence, VR developers must consider the users' physical environment when designing these applications. This makes it possible to match the real world with the virtual world (co-location), to increase the users' sensation of presence, and to inform users about their interaction capabilities (these capabilities are often limited by the size of the users' physical workspaces).

As an example, Figure 6.1 presents the viewpoints of three users collaborating in a multi-scale Collaborative Virtual Environments (msCVE): each user carries his interaction tools (instances of 2D Pointers/3D Rays [DF09] dedicated to collaborative interaction) within his Immersive Interactive Virtual Cabin (IIVC). This area, where the user can move and interact with co-located virtual objects, is represented as a colored virtual flying carpet. The IIVC also embeds a representation of its user's field of view. So each user can perceive the interactions limitations of the other users: he can see what they can reach with their virtual hand inside their IIVC, or what they can reach with a virtual ray inside their field of view.

Despite the fact that several existing VR applications take into account the features of the users' physical environment, such as for natural walking applications [CMRCL09] (see Figure 6.3), no consensus has been reached for these VR systems on how to embed this real environment into the virtual one. Each system models the real world in a particular way according to its requirements. We want to fill in the lack of standard metaphors for 3DUI to simplify Virtual Environment Design and Implementation [WL10]. We are interested in modeling the system, not only through device abstraction, but also by modeling the physical users' workspaces and their relationships.

We propose the Immersive Interactive Virtual Cabin as a generic model that enables VR developers to embed the users' physical environment into the Virtual Environment (VE) when designing or deploying new applications. This high-level model depicts the relationships between the real and the virtual world whatever the physical devices used or the room physical configuration. Our model deals with multi-scale collaborative virtual environments (msCVE as described in [ZF05]). This kind of virtual environment is more and more used to enable remote experts to work together on multi-scale structures such as scientific data or chemical molecules. Our model solves issues induced by collaborative sessions with remote users who interact from different physical environments: it enables users to perform a more effective collaboration by providing them a better understanding of the others' interaction capabilities, as it integrates users' physical workspaces and interaction tools in the virtual environment (see Figure 6.2).

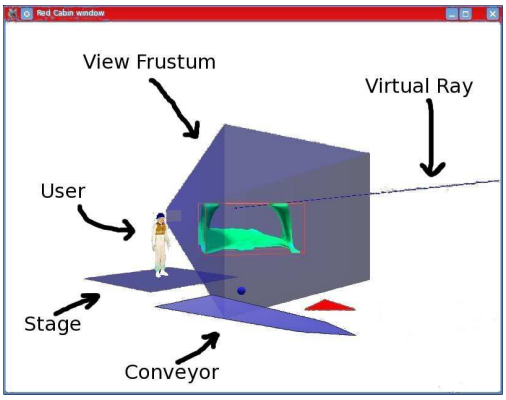


Figure 6.2: One user within his IIVC viewed by another user: we can see him, his conveyor, his stage, his view frustum and his virtual ray.

As user presence is a multi-sensory perception of the virtual world, we propose to model the users' physical environment as a structured hierarchy of sensory workspaces, such as motion, visual, sound, interaction or haptic workspaces. For example, a motion workspace is the area where a user can move his body. A visual workspace corresponds to a display device. A sound workspace represents the area in which a user perceives sound. An interaction workspace is the area where a user can interact. A haptic workspace is the area where a user can interact and have feedback when he uses a haptic device. The structured hierarchy of workspaces depicts the real-world spatial relationships between these workspaces. Our model also defines a set of operators to control each workspace, which enables VR developers to provide interactive features to end-users.

In this chapter, section 6.2 presents related work about the necessity of embedding the users' physical workspaces and about VR system design. Section 6.3 presents an overview of the IIVC concept, then section 6.4 describes the IIVC model, its structure and its operators. Section 6.5 describes the features it offers to end-users for interaction and collaboration, and how they can be implemented by VR developers. Section 6.6 illustrates how this model has been instantiated to design and implement Collaborative Virtual Environments (CVE), and how it could be used to design other existing VR techniques. Finally, section 6.7 concludes and gives some directions for future research on this topic.

6.2 Related work

Previous work about user interaction aims to provide users with a truly immersive experience. To achieve this, an increasing amount of this work must embed the users' physical workspaces into the VE to consider the users' interaction capabilities (see part 6.2.1). Although they often

propose interesting hierarchical data-structures, device abstractions and customization at run-time to adapt to physical devices, none of the existing software models take into account these physical workspaces in the VE when designing a new VR application (see section 6.2.2).

6.2.1 Embedding the physical workspaces into the VE

Embedding the user's motion workspace into the virtual environment offers the user an intuitive way to navigate by moving his own body. It also makes it possible to manage problems induced by the fact that the virtual world is often larger than this workspace. For example, the 3DM graphical modeler [BDHO92] enables a user to move on a "magic carpet" which represents the boundaries of the tracking area. The user uses Head Mounted Display (HMD), so he can perform real movements on the "magic carpet" to intuitively perform interactions. For long-distance navigation, he can also drive the "magic carpet" into the virtual world with a specific tool. The user reaches interaction tools through a 3D menu, which can be put on the "magic carpet" in order to travel with it. For natural walking in virtual worlds with a restricted workspace, the "Magic Barrier Tape" [CMRCL09] displays the boundaries of the physical workspace as a virtual barrier tape (see Figure 6.3). It informs the user about the boundaries of his walking workspace defined by the tracking area or the display devices. It also enables the user to intuitively navigate in the virtual world, without a break in presence, by "pushing" on the virtual barrier tape. Moreover, even if they do not display the user's motion workspace in the virtual environment, previous work about natural walking also has to consider these workspaces to prevent the user from colliding with the real environment or leaving the tracking area. Thus, they can determine when the user reaches the limits of the workspace to achieve redirected walking techniques [Raz05] or resetting techniques [WNR+07] such as the Freeze-backup or the 2:1 Turn.

Figure 6.3: The "Magic Barrier Tape" displays the limits of the user's mobility workspace. (Image from G. Cirio, IRISA.)

Additionally to the user's motion workspace, other workspaces (sound, visual, interaction and haptic, etc.) have to be considered in the virtual environment. For example, the "bubble" technique [DLB+05] proposes to display the limited workspace of a haptic device by a semi-transparent sphere that surrounds the manipulated cursor (see Figure 6.4). When the cursor is inside the "bubble", its motion is position-controlled. However, when the cursor is outside, the user can move the "bubble" into the virtual world using a velocity control. For prop-based haptic interaction [OC05], a real object (prop) is linked with a representation and an action in the virtual environment. So the haptic device workspace has to be embedded in the virtual environment to co-locate the prop with this virtual representation and to determine the user's interaction area. Moreover, the Hand Held Display (HHD) [Ams95] is a LCD display whose position and orientation are captured by a tracker. This display can be seen as a window on the virtual world. The system needs to know the

location of this visual workspace according to the user's location to compute the user's frustum. With the prop or the HHD, the co-location between the real world and the virtual world has to be maintained even if the user navigates or changes his scale in the virtual environment.

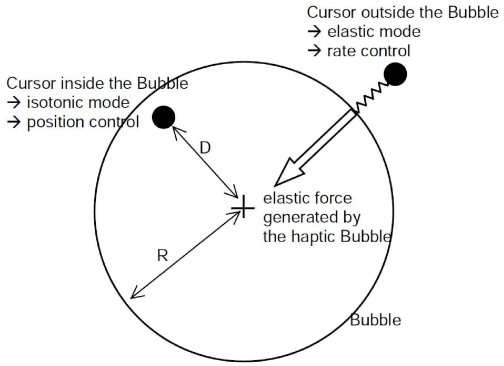


Figure 6.4: The "bubble" displays the limits of the haptic workspace. (Image from L. Dominjon, University of Angers.)

Within collaborative virtual environments, users must be able to communicate in order to perform closely coupled collaborative tasks. However, comprehension problems can occur for users with different viewpoints on the virtual world [FBHH99]. Even if they can see each other user's avatar, its position, and its orientation in the virtual world as in CALVIN [LJVD96], users have difficulty in perceiving what the others see, and more generally what they are doing and what they can do. To overcome these perception problems, Fraser et al. [FBHH99] explicitly outline each user's view frustum using a wireframe model. This model has to be adapted according to the users' display device. Moreover, when a user interacts with an object, they represent the link between this user and the manipulated object by extending the avatar's arm to the object. By extension, the *spatial model of interaction* proposed by [BBFG94] and implemented in Massive [GB95] can be seen as an interesting approach to describe users' multi-sensory perception. This spatial model defines sensory focus and nimbus for each user. The focus corresponds to the area in which a user has a sensory perception of the other users or of the virtual objects. The nimbus corresponds to the area in which the others have a sensory perception of this user. The focus and nimbus can have different sizes and shapes according to their corresponding media (sound, visual, haptic, etc.). These awareness areas are not necessarily symmetrical. Even if focus and nimbus do not represent users' real environment, they are directly linked to the features of this environment. Moreover, users carry their focus and nimbus when they move in the virtual world.

6.2.2 Software models for VR system design

Lots of VR software models have been proposed to support the development of virtual reality applications. Even if the first ones are very specific to particular technical features (scene-graph, operating system, input and output devices, etc.), some models such as VRJuggler [BJH+01] aim to be independent from the operating system and VR hardware configurations. VR^{2}S [SRH05a] can also support multi-sensory output and various input paradigms. This makes it possible to run VR applications in desktop environments by simulating VR devices with standard desktop input paradigms. Additionally to this device abstraction, rendering in VR^{2}S can be performed with several low-level APIs such as OpenGL or ray-tracing systems. As stated by Ohlenburg et al. [OBL07], these device abstraction layers provide a solution to deal with a large variety of interaction devices, but are usually limited to a specific set or type of input devices. So they introduce DEVAL as a generic device abstraction layer that defines device classes and structures them hierarchically. Thus, it can be easily extended by new device types. However, all these

abstractions consider devices only as input or output data, and not as real objects with their associated physical workspace to embed into the virtual world.

Robinett et al. [RH92] propose a hierarchy of coordinate systems to model a Head Mounted Display (HMD) system. This hierarchy of coordinate systems enables the system to modify the user position, orientation and scale in the virtual environment and to maintain the relationship between the real world and the virtual world using transform compositions. Dive [CH93, Hag96, FS98] defines also its own data-structure, which describes a scene in the manner of a scene-graph. Dive and Diverse [KAKS02] aims at creating extensible, reconfigurable and device independent virtual environments, through device abstraction and modular design. In this context, a program can run on different hardware configurations using appropriated configuration files. Dive stresses that VR applications should be adaptable to hardware and low-level software changes. In particular Dive proposes high-level concepts such as Vehicle, User and Body Abstraction [Ste08] in order to manage different hardware configurations. Simple Virtual Environment (SVE) [KBH00] also allows to design VR applications independently from the system configuration at run-time. Its design allows a variety of device configurations to be specified at run-time by providing a separation between the devices used and the environment model, and by defining how the device input affects the model. It also makes it possible to configure the software in order to use the best possible devices at run-time. SVE makes a separation between the VE model and the physical devices used, so it is easily configured according to the I/O devices used. The VE model includes a description of the 3D scene and a model of the user. This user model can be driven by input devices, and it can drive output devices. However, none of these VR software consider the physical devices as explicit 3D volumes that virtual representation could be embedded within the virtual world.

For collaboration, Zhang et al. [ZF05] propose a similar approach using the Java 3D scene-graph [SD99]. The Java 3D `ViewPlatform` concept uses a particular coordinate system to model the users' display device, with the idea of *"Write once, view everywhere"*. Even if this `ViewPlatform` makes it possible to adapt a VR application to several display devices, it cannot represent the users' visual workspace in the virtual environment. Moreover, it is not able to deal with other sensory workspaces.

Mulder et al. [MB04] describe a first notion of sensory workspaces. For a particular immersive device (a mirror-based display), they model the user's visual space and the user's interaction space. The visual space is defined according to user's head position in relation to the display device position. The interaction space, where a user can perform direct 3D interaction, is limited to the area that the user can reach. They define the user's "direct workspace" as the combination of these two spaces. Several of these personal immersive devices can be placed side by side to enable several users to collaborate in a "*physically shared* workspace". However, each user seems to be unable to freely navigate (i.e. to move his "direct workspace") in the virtual environment because the spatial relationship with the others has to be maintained. Moreover, this solution does not enable users to have remote collaboration and to visualize the others' "direct workspace" in the virtual world.

6.2.3 Synthesis

Many kinds of VR applications require the users' physical environment to be embedded into the virtual environment. This embedding aims to simply model or to represent this real environment into the virtual world. Modeling users' physical environment improves user presence by matching the virtual world with the real world and by providing an environment safe from collisions or tracking problems. Representing the boundaries of users' physical workspaces enables users to be aware of their interaction capabilities (or the interaction capabilities of the other users in the collaborative case). However, each VR application achieves a particular embedding of users' physical environment to meet its requirements instead of proposing a generic software model.

Some VR software models propose a device abstraction layer to enable developers to design

applications by simplifying the integration of various input or output devices. Moreover, they propose also to specify the devices configuration at runtime in order to adapt the software to hardware devices with no additional programming. However, they do not deal with the representation of these devices in the virtual environment, and they can neither describe the spatial relationships between these physical devices, nor model the users' physical workspace associated to each device.

Other solutions describe the organization of users' physical environment by a hierarchy of coordinate systems and introduce the notion of workspace, but they do not consider the physical workspaces of a user as explicit 3D volumes. Moreover, these approaches depend on the system properties such as the scene-graph, the system architecture, etc. Nevertheless, these can be seen as a first basic hierarchy of workspaces, even if they do not propose software models for embedding multi-sensory workspaces associated to various physical devices.

Last, the notion of workspaces introduced by Mulder et al. [MB04] must be generalized to all the sensory workspaces and to various devices. It must also maintain the spatial relationships between workspaces even if users navigate or change their scale in the virtual environment in order to combine several interaction techniques.

6.3 Overview of the IIVC

We need a generic solution that considers the users' physical environment during the VR software design, its deployment and its use. This solution must make the link between these three steps: it must propose a high-level model to describe, configure and modify the users' physical workspace organization whatever the immersive devices used.

6.3.1 The hierarchy of workspaces

We propose to model the users' physical environment as a structured hierarchy of virtual workspaces. We define the motion workspace as the area where a user can move his body. The visual workspace is not limited to a display device but to what the user can see through and around such a device. A sound workspace represents the area in which a user perceives sound. An interaction workspace is the area where a user can interact. A haptic workspace is the area where a user can interact and have feedback when he uses a haptic device. We call *stage* the reference workspace of our hierarchy (see section 6.4.1 for more details about this structure). Each virtual workspace must be described and located in relation to this *stage* or to another workspace included in the *stage*. Thus, we obtain a structured hierarchy of workspaces that depicts the real-world spatial relationships between these workspaces. Each workspace can contain real or virtual objects according to its sensory features, such as a tangible interface co-located with a virtual tool [OC05].

6.3.2 The IIVC concept

We propose the Immersive Interactive Virtual Cabin (IIVC) concept as a generic model to describe and manage the relationships between users, their physical environment, the virtual environment and the VR software developers. The IIVC is a link between the real world and the virtual world, but it can also be seen as a link between the end-users and the VR software developers (see Figure 6.5).

Coexistence End-users are located in the physical environment, so they can act on the real objects and on the input devices.

Design Developers create the virtual environment and choose which interaction and navigation techniques will be used.

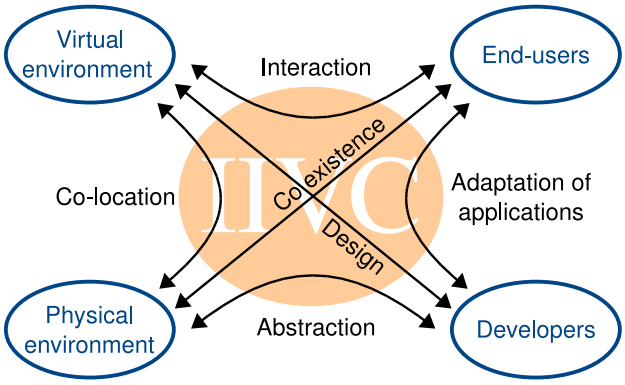


Figure 6.5: The IIVC concept.

Interaction End-users perform navigation and interaction tasks in the virtual environment. Sometimes, they have to perform these tasks in collaboration with other users. A good presence experience in the virtual environment enables them to interact more effectively.

Co-location 3D spaces of the virtual environment match 3D spaces of the physical environment to link real objects with their representation and action in the virtual world.

Abstraction Developers can design VR software with an abstraction of users' physical environment, which makes VR software more generic.

Adaption of applications Developers can efficiently configure and adapt applications to end-users' real environments.

6.4 The IIVC model

This section describes the structure and the main operators of the IIVC model.

6.4.1 The IIVC structure

The IIVC can be defined as an abstraction of the users' physical environment in the virtual world. It enables developers to implement their VR software without considering the physical devices used. For example, developers only have to manage position, orientation and scale of each user's IIVC when they develop navigation techniques. In a second step, each IIVC is configured with the features of each user's physical devices (size, shape, hierarchy of workspaces). The IIVC is based on three main components: the workspace, the *stage*, and the *conveyor*.

The *stage* is a virtual description of the users' real environment. It usually matches the room where users interact, but it is also the virtual space containing the virtual representations of users' workspaces. These workspaces are defined by the features of the physical devices used. For example, motion workspace limits are often defined by the boundaries of the area in which users can move: position of the display devices (such as in CAVE™ [CNSD93] or a Reality Center) or limits of the tracking area. These workspaces are organized in a hierarchy of included 3D spaces into the *stage*. Each workspace has its own 3D shape and its own coordinate system to locate smaller workspaces or objects (real or virtual) that it contains. The *stage* uses its own coordinate system to locate directly or indirectly all the users' workspaces and all the objects of the IIVC. With this organization, the IIVC model is able to deal with physical reconfiguration such as modifications of workspace position and shape, additions of new screens or other devices, etc.

The *conveyor* is the integration frame of the *stage* into the virtual world. This *conveyor* is located in the virtual world coordinate system, so it has its own position, orientation and scale in

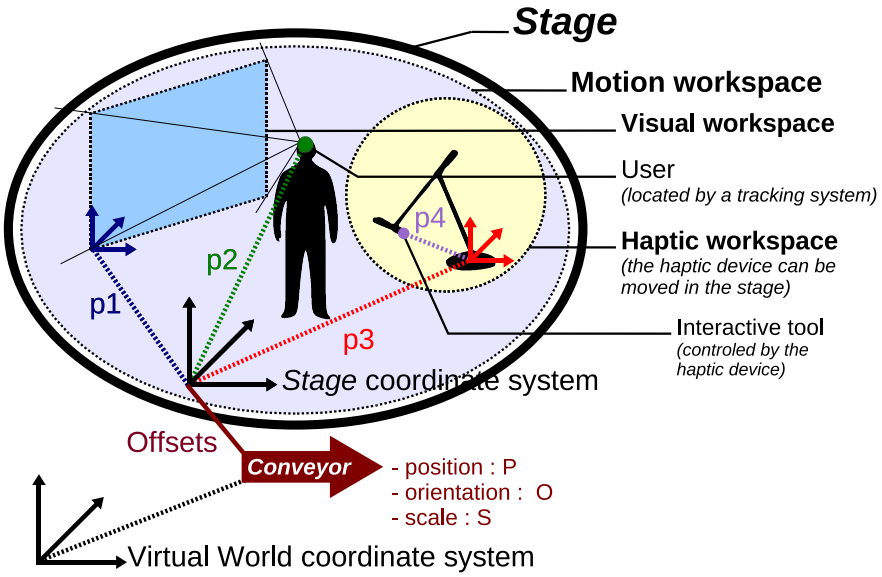


Figure 6.6: The IIVC structure: the *conveyor* carries the *stage* with its *workspaces* in the virtual world.

this world. The *stage* is linked to the *conveyor* with position, orientation, and scale offsets (see Figure 6.6). The *conveyor* also defines the navigation technique, the travel direction, the rotation center, and the scale of the IIVC. So the *stage*, its workspaces and consequently the objects inside the workspaces are carried by the *conveyor* when it moves or changes its scale in the virtual world.

The *conveyor* is totally virtual, while the *stage* makes the link between the real world and the virtual world. With this splitting into two parts, we have to decide where to put the limit between the *stage* and the *conveyor*. In other words, we have to choose which part of the real world must be embedded in the virtual environment. Indeed, we cannot represent all the real world in the virtual environment for all users. So we propose to define the limit of the *stage* as the last physical level which cannot move during the simulation. For example, in a CAVE™, the limit of the *stage* will be the cube defined by the screen's position. However, if the user interacts on a mobile platform such as a flight simulator, the limits of the *stage* will be the space surrounding the platform.

Like in Dive [FS98], we propose to manage our own data-structure in the manner of a scene-graph, without direct dependence to the 3D scene-graph that we use for our 3D graphic visualization. This data-structure rely upon the `SupportedObject` component: it is a `VirtualObject` that can be attached to a support, it can be compared to the famous `Transform` VRML Node. This generic component is provided with mechanisms ensuring the proper propagation of data updates between components at run-time, in order to make it able to compute its state relative to its support state. We use the `Observer` design pattern (GoF293)[Gam95] to propagate the changes from a support towards its supported objects.

The IIVC software architecture is based on a central component: the `Workspace` (see Figure 6.7). This component is an extension of the `SupportedObject` with new features described section 6.4.2. The `Workspace` is able to manage virtual objects such as virtual lights, virtual rays, virtual viewing frustums and to include other workspaces. The `Stage` is a particular `Workspace` that is linked to a `Conveyor`. The support of a `Stage` should always be a `Conveyor`, this is why we choose to make a special link between these two classes. The `Stage` is the root of hierarchy of the users' physical workspaces. Lastly, the `Conveyor` describes the navigation technique used.

All the virtual objects that will be embedded in the IIVC inherit also from the `SupportedObject`, such as the `User`, the `VirtualHand`, the `VirtualRay` or the `VirtualLight`. Each of these classes comes with its own behavior and features that we will not detail here.

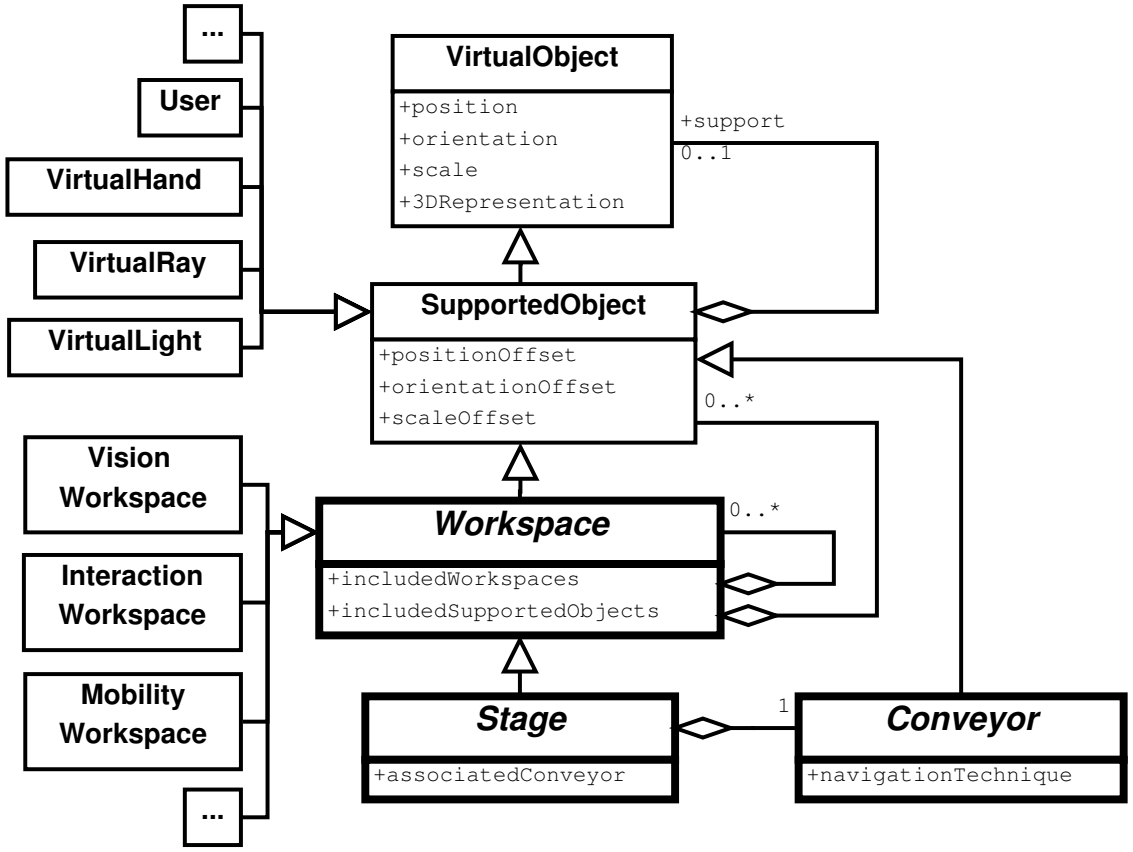


Figure 6.7: Partial UML model of the IIVC.

In the same way, specialized workspaces such as the `VisionWorkspace`, the `MobilityWorkspace` or the `InteractionWorkspace` inherit from the `Workspace` and come with their own behavior and features. For example such workspaces will require at least to know the relative location of their associated `User` in order to adapt the view to his position or to make him aware of some interaction possibilities or constraints (see Figures 6.8 and 6.9).

Last, the IIVC software architecture makes it possible to describe the users' physical environment independently from the usual 3D scene-graph or the 3D graphics API used to visualize it. It allows to switch from one 3D graphics API to another one without changing the core of our VR software, but only the component in charge of the coupling with the 3D visualization. This architecture can be described in a configuration file and its components can be associated to external 3D graphical representations. These graphical representations can be described using languages such as X3D or Collada without modifying the IIVC software components.

6.4.2 The IIVC operators

The operators are the basic and generic operations that are used to manage the IIVC structure. We provide a library that enables VR developers to implement several features as described in the section 6.5. First, the basic operators are:

Bo1: modify the position (6 DoF) or scale of a `VirtualObject`,

Bo2: modify the features of a `VirtualObject` (range of a virtual light or a virtual ray, etc.),

Bo3: provide a new support to a `SupportedObject`,

Bo4: modify the offset values of a `SupportedObject`,

Bo5: add or remove a `VirtualObject` into a workspace,

Bo6: provide a new `Conveyor` to a `Stage`,

Bo7: compute the local or global position of a `SupportedObject` in relation to another frame.

Second, we provide higher level operators, called "advanced operators" obtained through combination of basic operators, such as:

Ao1: superpose several `Stages` or several `Conveyors`,

Ao2: provide the same `Conveyor` as a support to several `Stages`,

Ao3: link a `Conveyor` to a `VirtualObject`,

Ao4: detect the proximity of `VirtualObjects`,

Ao5: compute the intersection of `Workspaces`,

Ao6: modify the shape of a `Workspace` (for example the virtual frustum associated to a `VisualWorkspaces`),

Ao7: restrain DoF for position modification.

This set of seven advanced operators does not pretend to cover all possible operations in a VR, it will have to be extended in the future.

6.5 The IIVC main features

The IIVC concept provides several features to VR application designers in order to optimize end-users' navigation, interaction, presence and collaboration according to their physical environment. We analyze the IIVC main features from both an end-user's point of view and a developer's point of view. The IIVC enables VR developers to integrate many VR features proposed in the literature, and also to introduce new VR features thanks to its particular architecture.

6.5.1 Navigating with the IIVC

Navigation from an end-user's point of view

Users can move within the motion workspace included in their *stage*. If a user can be located in this workspace (with a tracking system), his view frustum must be distorted according to his head position (head-tracking). This visual workspace can be seen as the *stage* "windows" on the virtual world. It enables users to observe or to position themselves in an intuitive way in the virtual environment.

Users can use almost any navigation technique or metaphor proposed in the literature in order to move their IIVC. For example, they can "fly", "walk", "teleport" themselves, turn around a position or an object, join or follow another user or another object, etc.

Navigation from a developer's point of view

Some of these navigation facilities, such as real or virtual walking, flying and teleportation, are provided by directly using some basic operators (**Bo1**, **Bo4**) of section 6.4.2.

Higher-level features are obtained by combining these operators. For example, allowing a user to select an object in order to turn around it, can be realized by providing this object as the new support of his conveyor (**Bo3**, **Ao3**) (with a null translation offset), computing the new translation offset of the stage according to the current positions of the virtual object, the conveyor and the stage (**Bo7**, **Bo4**), and restraining navigation interaction to only the rotation of the conveyor (**Ao7**). Joining or following a user or an object can be achieved in the same way.

6.5.2 Carrying 3D interaction tools

Interaction from an end-user's point of view

3D interaction tools such as a virtual ray or a virtual hand can be included in the users' workspace as particular real or virtual objects. So, as in 3DM [BDHO92], these interaction tools are automatically carried by the IIVC when it moves or changes its scale in the virtual world. Moreover, users can organize their workspaces by placing these interaction tools according to the kind of interaction they have to perform.

Interaction from a VR developer's point of view

It is easy for a VR developer to combine such interaction techniques with navigation, because he can locate these interaction tools in relation to the *stage* coordinate system of the users' workspaces (see Figure 6.6) (**Bo3**, **Bo4**).

6.5.3 Making users aware of the physical environment

Awareness from an end-user's point of view

A user and the real objects located in his physical environment can be embedded in the virtual world through the *stage*. So the user and these real objects can be co-located in the real and virtual world as in [OC05], even if the IIVC is moved or scaled in the virtual world.

An IIVC makes the user aware of his interaction capabilities and limitations by representing the limits of his workspaces (by a visual representation, by a sound, etc.). For example, it is possible to light up the virtual objects located in a user's interaction workspace in order to show him which objects are reachable (see Figure 6.8).

Virtual "semi-transparent" glasses can also prevent users from crossing the boundaries of the tracking system or colliding with the display device (in a full immersive device). When the user inside the immersive device is far from the limits of his motion workspace, the glasses are totally transparent. When the user comes closer to these limits, the glasses become visible to avoid the user crossing the workspace limits and breaking his presence (see Figure 6.9).

Awareness from a developer's point of view

Users' physical workspaces are limited by walls, boundaries of tracking systems, display devices, etc. By embedding these workspaces into the virtual environment as 3D volumes, the IIVC makes the virtual world match the real world, and it makes it possible to perform some computations, for example collision detection between users and workspaces (**Ao4**, **Ao5**), or virtual matching such as adjustment of the range of a virtual light to the interaction workspace geometry and size to highlight all accessible objects (**Bo2**, **Ao5**).

6.5.4 Collaborating through several IIVC

Collaboration from an end-user's point of view

IIVC navigation lets users progress independently from the other users in the virtual world: users can have heterogeneous perspectives of the virtual world, which can be very effective for collaborative work [BCF+08].

The IIVC provides features to improve the collaboration between users, such as joining or making collaborative navigation together, or interacting with the *conveyor* or the *stage* of another user in order to move, rotate, or scale it.

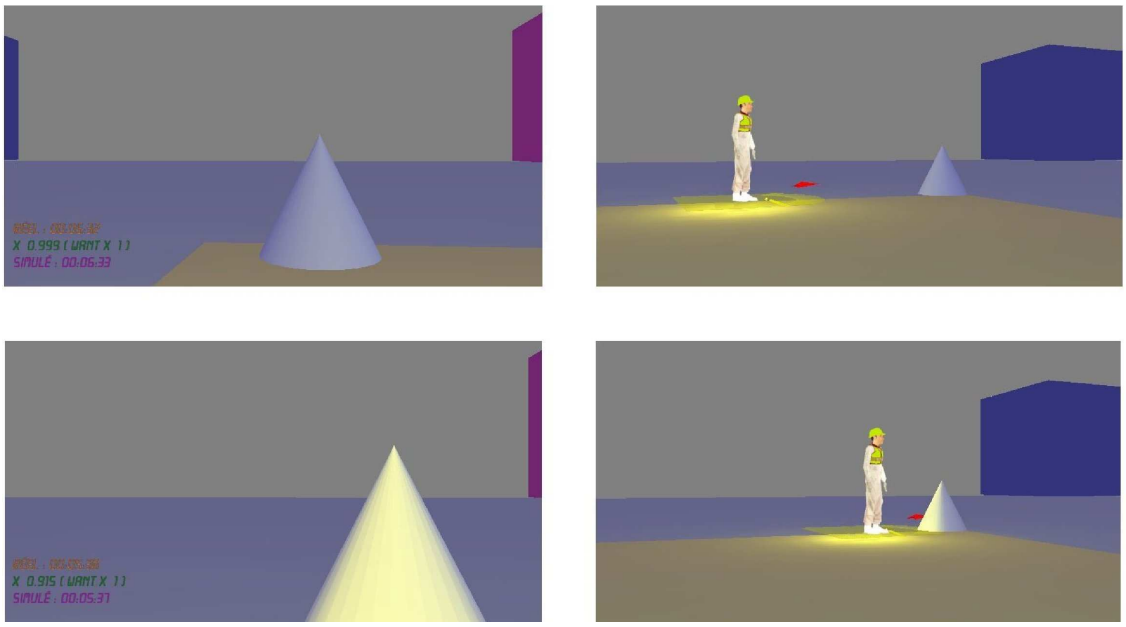


Figure 6.8: When objects enter the yellow user's interaction workspace, they are illuminated by a colored light (pictures on the left). The other users can also see that objects enter the yellow user's interaction workspace (pictures on the right).

Figure 6.9: The closer to the display device the user comes, the more visible the "semi-transparent" glasses become, to avoid collision with the physical workspace.

An IIVC can lap over another one: it establishes a relationship between different real environments through the virtual world. It has some restrictions because the real objects cannot appear in each real environment. However, a real object in an IIVC can have a virtual representation in the virtual world, so it can appear virtually in the other IIVC.

The IIVC represents users inside their physical devices in the virtual world in order to make users aware of the other users' interaction capabilities, which can improve the collaboration. It can help them to understand what the others are doing and where they are looking (see Figure 6.1), which physical device they are using, which object they can reach without performing a navigation task, etc. For example, Figure 6.8 shows the yellow user's interaction workspace lighting up to inform other users about which objects this user can reach.

Collaboration from a developer's point of view

As in Robinett et al. [RH92], when a user performs a navigation task, the VR developer can choose to move, rotate, or scale the IIVC rather than the virtual world, which allows each user to have

their own viewpoint (**Bo1**, **Bo4**).

Synchronization of several users can be realized by setting their *conveyor* at the same position (**Ao1**) or by linking their *stages* to the same *conveyor* (**Ao2**).

Considering that the IIVC represents its user's physical environment in the virtual environment, it can also be considered as an interactive object, which allows a VR developer to offer interaction with an IIVC of other users (**Bo1**, **Bo4**). It also naturally makes it possible to overlap several IIVC (**Ao1**) to provide collaborative awareness.

6.6 IIVC applications

The IIVC model has been implemented as a set of reusable modules, and all users' physical workspaces can be described within a file that we call the configuration file.

First, we present how we have used the IIVC to design several and implement multi-scale collaborative virtual environments. Then, we discuss how existing VR techniques could be designed using the IIVC model.

6.6.1 First instances

To demonstrate the possibilities of the IIVC, we have seamlessly integrated several classical interaction techniques in the IIVC such as virtual ray, and several collaboration facilities such as techniques to meet others, to navigate with the others or to leave 3D annotations (for example, a landmark representing an interactive viewpoint). All these tools can be carried by the user in his IIVC.

We thus obtained multi-scale collaborative virtual environments [DFNA08] that we have tested with simple workstations, with a Reality Center (an immersive device with stereoscopic vision, head-tracking and an area in which the user can move) and with a workbench. The IIVC model adapts our applications seamlessly to these kinds of devices: we have just to change the application configuration files. It also enables different users to interact in the same virtual environment with different devices: we have tested our collaborative applications with one user in a Reality Center and two other users in front of simple workstations (see Figure 6.1).

6.6.2 Instances of "state of the art" VR techniques

To illustrate the use of the IIVC model, we discuss how this model could be useful to design three "state of the art" VR techniques. These techniques involve a motion workspace for the first one, a haptic workspace for the second one, and a movable visual workspace for the last one.

Limited motion workspace

The "Magic Barrier Tape" [CMRCL09] (see Figure 6.3) could be implemented using the IIVC by displaying the virtual barrier tape just inside the real limits of the motion workspace, and this motion workspace would be directly included in the *stage*. As long as the user stays in this delimited area, he can freely walk to navigate in relation to the *stage* (**Bo4**). But, when he pushes on the virtual barrier tape, spatial movements of the *conveyor* would be computed from this action on the barrier tape (**Bo1**, **Ao7**). Thus the user could perform long-distance navigation in the whole virtual world by moving the *conveyor*.

Limited haptic workspace

The "bubble" technique [DLB+05] (see Figure 6.4) could be implemented using the IIVC by modeling the workspaces of the "bubble". The limited workspace of the haptic device would be rep-

resented by a concentric sphere that would be slightly smaller than this workspace. This haptic workspace would be included in the global motion workspace that would be also included in the *stage*. As long as the 3D cursor associated to the haptic device stays in the "bubble", it would be used for interaction within the *stage* as usual. But, when the cursor goes outside this area, it would be used for navigation: spatial movements would be computed from the cursor position in relation to the "bubble" boundaries (**Ao4**, **Ao5**). These movements can be used to move the *conveyor* (**Bo1**) in the virtual world in order to maintain the co-location of the "bubble" with the haptic device.

Movable visual workspace

The visual workspace associated to the screen of the Hand Held Display (HHD) [Ams95] could be defined as a movable workspace included in the user's motion workspace. As the user's head and this visual workspace would be located in the motion workspace, it would be easy to compute the visual workspace deformation (a viewing frustum) (**Ao6**) and the image to display on the screen. The way to compute this deformation would stay the same even if the user navigates by moving his *conveyor* and consequently the whole IIVC in the virtual world.

6.7 Conclusion and future work

With its ability to manage a hierarchy of 3D workspaces, the Immersive Interactive Virtual Cabin (IIVC) provides a generic software model to embed users' physical workspaces in a virtual environment. The IIVC is an abstraction of immersive devices which enables the VR developers to design applications without taking into consideration which immersive devices will be used. It can be adapted to a simple workstation or to a full immersive device such as a CAVE™. It matches the real world with the virtual world to maintain head-tracking of users or co-location of real objects even if users navigate in the virtual world. This navigation (position, orientation and scale changes) is independently performed by each user and is also applied to the interaction tools included in the users' workspaces.

The IIVC software architecture makes it possible to describe the users' physical environment independently from the 3D graphics API used to visualize it. Only one component is in charge of the coupling with the 3D visualization, through a configuration file to associate its components to external 3D graphical representations. The IIVC is useful to Collaborative Virtual Environments (CVE) developers because it automatically provides a 3D representation of a user's physical workspaces to the other users who share the same CVE, making them naturally aware of the physical activity and limitations of each other user.

As we have essentially used visual and motion workspaces, future work should consist now in exploring other kinds of workspaces such as sound or haptic workspaces.

We will also need to enhance our existing model with other advanced operators, especially for the perception of the users' interaction capabilities and for collaboration. We will have to evaluate how much embedding users' physical workspaces within the IIVC can enable the user to better understand their interaction capabilities or limitations. In collaborative situations, it should naturally provide a better awareness of the other users' activities and interaction capabilities.

Finally, we should propose a standardized formalism to describe workspace management and manipulation. Thus, a language to describe the physical workspace of each user and its mapping with the virtual environment should be defined. It could be an extension to X3D or Collada, or to 3DFC, as for describing interactive and collaborative features in chapter 4.

Conclusion and Perspectives

Conclusion

We have proposed some cues to answer the challenges of designing dynamic CVE to meet their architectural requirements at a system and at a software level:

- a new adaptive data distribution model that enables a CVE system to achieve three data distribution modes to deal with several applications requirements and various kinds of network connections. The data distribution mode can be individually chosen for each object according to the function that it fulfills in the virtual environment. Moreover, this data distribution can be dynamically changed during a session to adapt itself to the tasks that users need to perform in the CVE.

- mechanisms to detect and visualize network problems while interacting within a networked virtual environment, which make it possible to perform a synchronization by groups of users to ensure the best trade-off between synchronization and interaction latency.

- PAC-C3D, a new architectural model dedicated to 3D collaborative virtual environments. It can deal with the main distribution modes encountered in CVE, and it makes it possible to design a CVE with very small dependency on a 3D graphics API, providing easy interoperability between 3D graphics API.

We have also addressed the design of collaborative interactions and the design of what can contribute to give users a better awareness of the collaboration and of their environment:

- a new formalism for describing 3D interactions in virtual environments. It defines what a virtual interactive object and an interaction tool are, and how these two kinds of objects can communicate together. It is a first step toward a description language to describe the interactive and collaborative properties of virtual objects.

- new metaphors for interaction within Collaborative Virtual Environments (CVE). They are the evolution of usual 2D or 3D interaction metaphors that have been adapted to fit with collaborative interactions, and to make users aware of this collaboration.

- the Immersive Interactive Virtual Cabin (IIVC), which is an abstraction of immersive devices which enables the VR developers to design applications without taking into consideration which immersive devices will be used. It makes it possible to describe the users' physical environment independently from the 3D graphics API used to visualize it, and also to visualize the physical features of the hardware input and output devices.

All this work has been realized in the context of 5 Master thesis (Michaël Rouillé, Aurélien Fénals, Sébastien Thomas, Ting-Yun Lu and Cédric Fleury) and 5 PhD thesis (Chadi Zammar, Laurent Aguerreche, Cédric Fleury, Rozenn Bouville and Thi Thuong Huyen Nguyen).

Perspectives

Frameworks for collaborative virtual environments are now mature enough to allow researchers to focus on higher-level description of collaboration rather than on low-level system features. Nevertheless, there is still some very interesting work to be done in order to enhance interoperability between rendering engines (graphics engines, physics engines, behavior engines, collaboration engines, interaction engines, ...), by describing at a high level of abstraction what are the data that must be exchanged between these engines. The PAC-C3D architectural model implemented in the Collaviz framework and the Scene Graph Adapter are some steps toward this high-level interoperability, but it is still a work in progress.

Another important topic is to let CVE designers focus on high-level collaboration rather than on low-level distribution or synchronization features. We have contributed to this topic through the OpenMASK and Collaviz frameworks, which propose to describe a shared virtual universe through virtual objects, without needing to worry about how the distribution and the synchronization are achieved, only focusing upon interactive and collaborative features. For now, these descriptions are made through dedicated (xml-based) description languages, and a first attempt has been made to extend the Collada description language. An effort should be made to propose an official description of shared virtual objects that could be an extension to X3D or Collada, or to 3DFC.

CVE designers who want to propose new 3D interaction metaphors for OpenMASK and Collaviz still have to code some behavior (in C++ or Java). We should go one step beyond this coding phase by defining a Domain Specific Language (DSL) dedicated to interaction, that would be some kind of Model Driven Engineering (MDE) development tool, able to produce C++ OpenMASK code or Collaviz Java code, or any other framework-dedicated code.

Establishing as automatically as possible a good matching between the virtual environment and the physical environment of the end-users is still a challenge. Our IIVC concept is a first answer to this problem, it can be improved by providing more high-level operators and by generalizing this approach to other frameworks than OpenMASK and Collaviz. Once again, a MDE approach would certainly be very interesting, with a DSL for describing the features of the physical environment and their matching with virtual objects of the virtual universe.

Last, we still have to improve the collaboration between distant users who are sharing a virtual environment, by proposing again and again more efficient metaphors for 3D collaborative interactions. This topic is still very relevant: it was the main subject of the 3DUI 2012 contest. We participated to this contest by proposing some solutions based on the Collaviz framework in order to enhance collaboration between two users [NFD12] and we will go on proposing new solutions dedicated to fully immersive collaboration.

Bibliography

[ADA09a] Laurent Aguerreche, Thierry Duval, and Bruno Arnaldi. A description of a Dialog to Enable Interaction between Interaction Tools and 3D Objects in Collaborative Virtual Environments. In *Proc. of VRIC 2009*, pages 63–73, Laval, France, April 2009.

[ADA09b] Laurent Aguerreche, Thierry Duval, and Bruno Arnaldi. Analyse de techniques de coopération en environnements virtuels 3D. *Revue Technique et Science Informatiques (TSI)*, 28(6-7):767–797, 2009.

[ADL09] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. 3-Hand Manipulation of Virtual Objects. In *Proc. of JVRC 2009 (Joint Virtual Reality Conference of EGVE - ICAT - EuroVR)*, pages 153–156, 2009.

[ADL10a] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. Comparison of Three Interactive Techniques for Collaborative Manipulation of Objects in Virtual Reality. In *Proc. of CGI*, 2010.

[ADL10b] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. Reconfigurable Tangible Devices for 3D Virtual Object Manipulation by Single or Multiple Users. In *Proc. of VRST*, pages 227–230, New York, NY, USA, 2010. ACM.

[ADL11] Laurent Aguerreche, Thierry Duval, and Anatole Lécuyer. Evaluation of a Reconfigurable Tangible Device for Collaborative Manipulation of Objects in Virtual Reality. In *Proc. of UK Eurographics Chapter, Theory and Practice of Computer Graphics*, pages 81–88, Warwick, United Kingdom, September 2011.

[AFM+99] David Anderson, James L. Frankel, Joe Marks, Darren Leigh, Eddie Sullivan, Jonathan Yedidia, and Kathy Ryall. Building Virtual Structures with Physical Blocks. In *Proc. of UIST*, pages 71–72, 1999.

[Agu10] Laurent Aguerreche. *Partage d'interactions en environnements virtuels : de nouvelles techniques collaboratives basées sur un protocole de dialogue générique.* These, INSA de Rennes, June 2010.

[AHV04] C. Anthes, P. Heinzlreiter, and J. Volkert. An Adaptive Network Architecture for Close-Coupled Collaboration in Distributed Virtual Environments. In *VRCAI'04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 382–385, New York, NY, USA, 2004. ACM.

[Ams95] Denis Amselem. "A Window on Shared Virtual Environments". *Presence: Teleop. & Virtual Env.*, 4(2):130–145, 1995.

[ART] A.R.T. Gmbh website. `http://www.ar-tracking.de/`.

[BBDRA11] Rozenn Bouville Berthelot, Thierry Duval, Jérôme Royan, and Bruno Arnaldi. Improving Reusability of Assets for Virtual Worlds while Preserving 3D Formats Features. *JVWR (Journal for Virtual World Research)*, 4(3), November 2011.

[BBFG94] Steve Benford, John Bowers, Lennart E. Fahlén, and Chris Greenhalgh. "Managing Mutual Awareness in Collaborative Virtual Environments". In *Proc. of the Symp. on Virtual Reality Software and Technology*, pages 223–236, 1994.

[BBH+90] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. "Reality built for two: a virtual reality tool". In *SI3D'90: Proceedings of the symposium on Interactive 3D graphics*, pages 35–36, New York, NY, USA, 1990. ACM.

[BBRDA11] Rozenn Bouville Berthelot, Jérôme Royan, Thierry Duval, and Bruno Arnaldi. Scene Graph Adapter: An efficient Architecture to Improve Interoperability between 3D Formats and 3D Application Engines. In ACM, editor, *Web3D 2011 (16th International Conference on 3D Web technology)*, pages 21–29, Paris, France, June 2011.

[BBRDA12] Rozenn Bouville Berthelot, Jérôme Royan, Thierry Duval, and Bruno Arnaldi. 3DFC: a new container for 3D file formats compositing. In *Web3D 2012 (17th International Conference on 3D Web technology)*, pages 27–35, Los Angeles, United States, August 2012. ACM.

[BCF+08] D.A. Bowman, S. Coquillart, B. Froehlich, M. Hirose, Y. Kitamura, K. Kiyokawa, and W. Stuerzlinger. 3D User Interfaces: New Directions and Perspectives. *Computer Graphics and Applications, IEEE*, 28(6):20–36, 2008.

[BDHO92] Jeff Butterworth, Andrew Davidson, Stephen Hench, and Marc. T. Olano. "3DM: A Three Dimensional Modeler using a Head-Mounted Display". In *Proc. of the Symp. on Interactive 3D graphics*, pages 135–138, 1992.

[BGRP01] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock. Collaborative Virtual Environments. *Communication of the ACM*, 44(7):79–85, 2001.

[BH95] Doug A. Bowman and Larry F. Hodges. User interface constraints for immersive virtual environment applications. Technical report, Graphics, Visualization, and Usability Center GIT-GVU-95-26, 1995.

[BH97] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–ff., New York, NY, USA, 1997. ACM Press.

[BHB08] W. Broll, J. Herling, and L. Blum. Interactive bits: Prototyping of mixed reality applications and interaction techniques through visual programming. In *Proceedings of the 2008 IEEE Symposium on 3D User Interfaces*, 3DUI '08, pages 109–115, Washington, DC, USA, 2008. IEEE Computer Society.

[BHSS00] Cagatay Basdogan, Chih-Hao Ho, Mandayam A. Srinivasan, and Mel Slater. An experimental study on the role of touch in shared virtual environments. *ACM Trans. Comput.-Hum. Interact.*, 7(4):443–460, December 2000.

[BJH99] Doug A. Bowman, Donald B. Johnson, and Larry F. Hodges. Testbed evaluation of virtual environment interaction techniques. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '99, pages 26–33, New York, NY, USA, 1999. ACM.

[BJH+01] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. "VR Juggler: A Virtual Platform for Virtual Reality Application Development". In *Proc. of the IEEE Virtual Reality Conference*, pages 89–96, 2001.

[BKLP04] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley, 2004.

[BLO+05] Wolfgang Broll, Irma Lindt, Jan Ohlenburg, Iris Herbst, Michael Wittkamper, and Thomas Novotny. An infrastructure for realizing custom-tailored augmented reality user interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):722–733, November 2005.

[Bul] Bullet Physics Library website. `http://www.bulletphysics.com/`.

[CCN97] Gaëlle Calvary, Joëlle Coutaz, and Laurence Nigay. From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. In *Proceedings of CHI 97, ACM publ*, pages 242–249, 1997.

[CDG+93] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. "The SIMNET virtual world architecture". *IEEE Virtual Reality Annual International Symposium*, pages 450–455, Sep 1993.

[CFH97] Lawrence D. Cutler, Bernd Fröhlich, and Pat Hanrahan. Two-Handed Direct Manipulation on the Responsive Workbench. In *Proceedings of SI3D'97*, pages 107–114. ACM, 1997.

[CH93] Carlsson C. and Hagsang O. Dive – a platform for multi–user virtual environnement. *Computer and Graphics*, pages 663–669, 1993.

[CMRCL09] Gabriel Cirio, Maud Marchal, Tony Regia-Corte, and Anatole Lécuyer. "The Magic Barrier Tape: A Novel Metaphor for Infinite Navigation in Virtual Worlds with a Restricted Walking Workspace". In *Proc. of the 16th Symp. on Virtual Reality Software and Technology*, pages 155–162, 2009.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proceedings of SIGGRAPH'93*, pages 135–142, New York, NY, USA, 1993. ACM.

[Col] The Collaviz website. http://www.collaviz.org/.

[Cou87] Joëlle Coutaz. PAC: An Object Oriented Model for Implementing User Interfaces. *SIGCHI Bull.*, 19(2):37–41, 1987.

[Dav01] Malcolm Davis. Struts, an open-source MVC implementation. http://www.ibm.com/developerworks/library/j-struts/, february 2001.

[DDF+10] Florent Dupont, Thierry Duval, Cédric Fleury, Julien Forest, Valérie Gouranton, Pierre Lando, Thibaud Laurent, Guillaume Lavoué, and Alban Schmutz. Collaborative Scientific Visualization: The COLLAVIZ Framework. In *JVRC Demos*, 2010.

[DDS+99] Cédric Dumas, Samuel Degrande, Grégory Saugis, Christophe Chaillou, Mary-Luce Viaud, and Patricia Plénacoste. SpIn: a 3D Interface for Cooperative Work. *Virtual Reality Society Journal*, 1999.

[Dew99] Prasun Dewan. Architectures for Collaborative Applications. *Trends in Software, special issue on Collaborative Systems*, pages 169–193, 1999.

[DF02] Thierry Duval and Aurélien Fenals. Faciliter la perception de l'interaction lors de manipulations coopératives simultanées en environnements virtuels 3d. In *Informal communication in annex of the Proceedings of IHM 2002*, pages 29–32, 2002.

[DF09] Thierry Duval and Cédric Fleury. An asymmetric 2d pointer/3d ray for 3d interaction within collaborative virtual environments. In *Web3D'09: Proceedings of the 14th International Conference on 3D Web Technology*, pages 33–41, New York, NY, USA, 2009. ACM.

[DFNA08] Thierry Duval, Cédric Fleury, Bernard Nouailhas, and Laurent Aguerreche. "Collaborative Exploration of 3D Scientific Data". In *VRST'08: Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, pages 303–304, New York, NY, USA, 2008. ACM.

[DH02] J. Döllner and K. Hinrichs. A generic rendering system. *IEEE Transactions on Visualization and Computer Graphics*, pages 99–118, 2002.

[DLB+05] Lionel Dominjon, Anatole Lécuyer, Jean-Marie Burkhardt, Guillermo Andrade-Barroso, and Simon Richir. "The "Bubble" Technique: Interacting with Large Virtual Environments Using Haptic Devices with Limited Workspace". In *Proc. of the World Haptics Conference*, pages 639–640, 2005.

[DLT04] Thierry Duval and Christian Le Tenier. Interactions 3D coopératives sur des objets techniques avec OpenMASK. *Mécaniques et Industries*, 5(2):129–137, 2004.

[DLT06] Thierry Duval, Anatole Lecuyer, and Sebastien Thomas. SkeweR: a 3D Interaction Technique for 2-User Collaborative Manipulation of Objects in Virtual Environments. In *Proceedings of 3DUI'06*, pages 69–72. IEEE, 2006.

[DM00a] Thierry Duval and David Margery. Building Objects and Interactors for Collaborative Interactions with GASP. In *CVE 2000*, pages 129–138, San Francisco, United States, september 2000.

[DM00b] Thierry Duval and David Margery. Using GASP for Collaborative Interactions within 3D Virtual Worlds. In *Proceedings of the Second International Conference on Virtual Worlds (VW'2000)*, pages 65–76, Paris, France, july 2000. Springer LNCS/AI.

[DMR+97] Thierry Duval, Serge Morvan, Patrick Reignier, Fabrice Harrouet, and Jacques Tisseau. Arévi : une boîte à outils 3d pour des applications coopératives. *Numéro spécial de la revue Calculateurs Parallèles (coopération)*, pages 239–250, juillet 1997.

[DRC+00] Thierry Duval, Jordi Regincòs, Alain Chauffaut, David Margery, and Bruno Arnaldi. Interactions collectives locales en immersion dans des univers virtuels 3d avec gasp. In *ERGO-IHM*, October 2000.

[DWM06a] D. Delaney, T. Ward, and S. McLoone. "On consistency and network latency in distributed interactive applications: A survey – part I". *Presence: Teleoperators and Virtual Environments*, 15(2):218–234, 2006.

[DWM06b] D. Delaney, T. Ward, and S. McLoone. "On Consistency and Network Latency in Distributed Interactive Applications: A Survey – Part II". *Presence: Teleoperators and Virtual Environments*, 15(4):465–482, 2006.

[DZ06a] Thierry Duval and Chadi el Zammar. A migration mechanism to manage network troubles while interacting within collaborative virtual environments. In *Proc. of Virtual reality continuum and its applications - VRCIA*, pages 417–420, 2006.

[DZ06b] Thierry Duval and Chadi el Zammar. Managing Network Troubles while Interacting within Collaborative Virtual Environments. *CSAC'2006, Paphos, Cyprus*, pages 85–94, 2006.

[Eck07] Robert Eckstein. Java SE Application Design With MVC. http://www.oracle.com/technetwork/articles/javase/mvc-136693.html, march 2007.

[EPO95] Chris Esposito, W. Bradford Paley, and JueyChong Ong. Of Mice and Monkeys: A Specialized Input Device for Virtual Body Animation. In *Proc. of I3D*, pages 109–114, 1995.

[EW94] Clarence Ellis and Jacques Wainer. A conceptual model of groupware. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 79–88, New York, NY, USA, 1994. ACM.

[FBHH99] Mike Fraser, Steve Benford, Jon Hindmarsh, and Christian Heathq. "Supporting Awareness and Interaction through Collaborative Virtual Interfaces". In *Proc. of the 12th Symp. on User Interface Software and Technology*, pages 27–36, 1999.

[FCD+11] Cédric Fleury, Alain Chauffaut, Thierry Duval, Valérie Gouranton, and Bruno Arnaldi. A Generic Model for Embedding Users' Physical Workspaces into Multi-Scale Collaborative Virtual Environments. In *Proc. of ICAT*, pages 1–8, 2011.

[FDGA10a] Cédric Fleury, Thierry Duval, Valérie Gouranton, and Bruno Arnaldi. A New Adaptive Data Distribution Model for Consistency Maintenance in Collaborative Virtual Environments. In *Proc. of JVRC*, pages 29–36, 2010.

[FDGA10b] Cédric Fleury, Thierry Duval, Valérie Gouranton, and Bruno Arnaldi. "Architectures and Mechanisms to efficiently Maintain Consistency in Collaborative Virtual Environments". In *Proc. of Software Engineering and Architectures for Realtime Interactive Systems - SEARIS*, pages 87–94, 2010.

[FDGS12] Cédric Fleury, Thierry Duval, Valérie Gouranton, and Anthony Steed. Evaluation of Remote Collaborative Manipulation for Scientific Data Analysis. In *VRST 2012 - 18th Symposium on Virtual Reality Software and Technology*, Toronto, Canada, December 2012. ACM.

[FGH02] Pablo Figueroa, Mark Green, and H. James Hoover. Intml: a description language for vr applications. In *Proceedings of the seventh international conference on 3D Web technology*, Web3D '02, pages 53–58, New York, NY, USA, 2002. ACM.

[FGV+00] Fraser, M., Glover, T., Vaghi, I., Benford, S., Greenhalgh, C., Hindmarch, J., and Heath, C. Revealing the Realities of Collaborative Virtual Reality. In *Proceedings of CVE'2000, San Francisco*, pages 29–37, 2000.

[FGW01] Pablo Figueroa, Mark Green, and B. Watson. A framework for 3d interaction techniques. *CAD/Graphic.*, 8:22–24, 2001.

[FHZ96] Andrew Forsberg, Kenneth Herndon, and Robert Zeleznik. Aperture based selection for immersive virtual environments. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 95–96, New York, NY, USA, 1996. ACM.

[FN98] Emmanuel Frécon and Anneli Avatare Nöu. Building distributed virtual environments to support collaborative work. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '98, pages 105–113, New York, NY, USA, 1998. ACM.

[Fra95] John H. Frazer. *An evolutionary architecture*. Architectural Association, London, 1995.

[FS98] E. Frécon and M. Stenius. "DIVE : A scaleable network architecture for distributed virtual environments". *Distrib. Syst. Engng.*, 5:91–100, 1998.

[Fun95] T. A. Funkhouser. "RING: a client-server system for multi-user virtual environments". In *SI3D'95: Proc. of the symposium on Interactive 3D graphics*, pages 85–93, New York, NY, USA, 1995. ACM.

[Gam95] Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J. *Design patterns: Elements of reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GAW09] I. J. Grimstead, N. J. Avis, and D. W. Walker. "RAVE: the resource-aware visualization environment". *Concurrency and Computation : Pract. & Exper.*, 21(4):415–448, 2009.

[GB95] C. Greenhalgh and S. Benford. Massive: a distributed virtual reality system incorporating spatial trading. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, ICDCS '95, pages 27–, Washington, DC, USA, 1995. IEEE Computer Society.

[GG98] Gutwin, C. and Greenberg, S. Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness. *CSCW'98, Seattle, Washington, US*, pages 207–216, 1998.

[GG99] Carl Gutwin and Saul Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, 6(3):243–281, September 1999.

[GMMG08] Arturo S. García, José P. Molina, Diego Martínez, and Pascual González. Enhancing Collaborative Manipulation through the Use of Feedback and Awareness in CVEs. In *Proceedings of VRCAI'08*, pages 1–5. ACM, 2008.

[Gol90] Adele Goldberg. Information models, views, and controllers. *Dr. Dobb's J.*, 15:54–61, May 1990.

[GPS00] Chris Greenhalgh, Jim Purbrick, and Dave Snowdon. Inside massive-3: flexible support for data consistency and world structuring. In *Proceedings of the third international conference on Collaborative virtual environments*, CVE '00, pages 119–127, New York, NY, USA, 2000. ACM.

[Hag96] Olof Hagsand. "Interactive Multiuser VEs in the DIVE System". *IEEE MultiMedia*, 3(1):30–39, 1996.

[Han97] Chris Hand. A Survey of 3D Interaction Techniques. *Computer Graphics Forum*, 16(5):269–281, 1997.

[Hin00] J. D.K Hinrichs. A generalized scene graph. *Vision, modeling, and visualization 2000: proceedings: November 22-24, 2000, Saarbrücken, Germany*, page 247, 2000.

[HPGK94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 452–458, 1994.

[HPPK98] Ken Hinckley, Randy Pausch, Dennis Proffitt, and Neal F. Kassell. Two-handed Virtual Manipulation. *ACM Transactions on Computer-Human Interaction*, 5(3):260–302, 1998.

[HTP+97] Ken Hinckley, Joe Tullio, Randy Pausch, Dennis Proffitt, and Neal Kassell. Usability analysis of 3d rotation techniques. In *Proc. of UIST*, pages 1–10, 1997.

[IMWB01] Brent Edward Insko, Michael J. Meehan, Mary C. Whitton, and Frederic P. Brooks. Passive haptics significantly enhances virtual environments. Technical report, The University of North Carolina at Chapel Hill, 2001.

[ISA01] ISAR2001. Design of a component-based augmented reality framework. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, ISAR '01, pages 45–, Washington, DC, USA, 2001. IEEE Computer Society.

[IU97] Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proc. of CHI*, pages 234–241, 1997.

[Jav] The Java3D website. http://java3d.java.net/. Accessed August 22, 2012.

[JBu] The JBullet website. http://jbullet.advel.cz/.

[JDGMT04] C. Joslin, T. Di Giacomo, and N. Magnenat-Thalmann. "Collaborative virtual environments: from birth to standardization". *IEEE Communications Magazine*, 42(4):28–33, Apr 2004.

[Jef85] D. R. Jefferson. Virtual time. *ACM Trans. on Programming Language and Systems*, 7(3):404–425, 1985.

[JFM+08] S. Jourdain, J. Forest, C. Mouton, B. Nouailhas, G. Moniot, F. Kolb, S. Chabridon, M. Simatic, Z. Abid, and L. Mallet. "ShareX3D, a scientific collaborative 3D viewer over HTTP". In *Web3D'08: Proceedings of the 13th international symposium on 3D web technology*, pages 35–41, New York, NY, USA, 2008. ACM.

[JMo] The JMonkey website. http://jmonkeyengine.org/.

[jRe] The jReality website. http://www3.math.tu-berlin.de/jreality/.

[KAKS02] John Kelso, Lance E. Arsenault, Ronald D. Kriz, and Steven G. Satterfield. "DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments". *Virtual Reality Conference, IEEE*, 0:183, 2002.

[Kaz96] Kazman, R. Load Balancing, Latency Management and Separation of Concerns in a Distributed Virtual World. *Parallel Computations - Paradigms and Applications*, 1996.

[KBH00] G. Drew Kessler, Doug A. Bowman, and Larry F. Hodges. "The Simple Virtual Environment Library: An Extensible Framework for Building VE Applications". *Presence: Teleoper. Virtual Environ.*, 9(2):187–208, 2000.

[KT98] Marcelo Kallmann and Daniel Thalmann. Modeling objects for interaction tasks. In *Proceedings of the 9th Eurographics Workshop on Animation and Simulation (EGCAS)*, pages 73–86, Lisbon, Portugal, 1998.

[LCAA08] Xavier Larrodé, Benoît Chanclou, Laurent Aguerreche, and Bruno Arnaldi. OpenMASK: an Open-Source platform for Virtual Reality. In *IEEE VR workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, Reno, États-Unis, March 2008.

[LGH98] Vali Lalioti, Christophe Garcia, and Frank Hasenbrink. Virtual meeting in cyberstage. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '98, pages 205–212, New York, NY, USA, 1998. ACM.

[LJD97] J. Leigh, A. E. Johnson, and T. A. DeFanti. "Cavern: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments". *Virtual Reality: Research, Development, and Applications*, 2(2):217–237, 1997.

[LJVD96] J. Leigh, A.E. Johnson, C.A. Vasilakis, and T.A. DeFanti. Multi-perspective Collaborative Design in Persistent Networked Virtual Environments. In *Proceedings of VRAIS'96*, pages 253–260, 1996.

[LLHL07] D. Lee, M. Lim, S. Han, and K. Lee. "ATLAS: A Scalable Network Framework for Distributed Virtual Environments". *Presence: Teleoperators and Virtual Environments*, 16(2):125–156, 2007.

[LN02] Yann Laurillau and Laurence Nigay. Clover architecture for groupware. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 236–245. ACM, 2002.

[LPI07] Vincent LeClerc, Amanda Parkes, and Hiroshi Ishii. Senspectra: A computationally augmented physical modeling toolkit for sensing and visualization of structural strain. In *Proc. of CHI*, pages 801–804, 2007.

[MAC+02] David Margery, Bruno Arnaldi, Alain Chauffaut, Stéphane Donikian, and Thierry Duval. "OpenMASK: Multi-Threaded or Modular Animation and Simulation Kernel or Kit : a General Introduction". In *Virtual Reality International Conference (VRIC 2002)*, pages 101–110, 2002.

[MAP99] D. Margery, B. Arnaldi, and N. Plouzeau. "A General Framework for Cooperative Manipulation in Virtual Environments". In *Virtual Environments'99*, pages 169–178, 1999.

[MB04] Jurriaan D. Mulder and Breght R. Boschker. "A Modular System for Collaborative Desktop VR/AR with a Shared Workspace". *Proc. of the IEEE Virtual Reality Conference*, 0:75, 2004.

[MZ97] M.R. Macedonia and M.J. Zyda. "A taxonomy for networked virtual environments". *IEEE Multimedia*, 4(1):48–56, Jan-Mar 1997.

[MZP+94] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. "NPSNET: A network software architecture for large scale virtual environments". *Presence*, 3(4):265–287, 1994.

[NC91] L. Nigay and J. Coutaz. Building User Interfaces: Organizing Software Agents. In *Proceedings of Esprit'91*, pages 709–717, 1991.

[NFD12] Thi Thuong Huyen Nguyen, Cédric Fleury, and Thierry Duval. Collaborative Exploration in a Multi-Scale Shared Virtual Environment. In *3DUI 2012*, Orange County, United States, March 2012.

[NLSG03] Martin Naef, Edouard Lamboray, Oliver Staadt, and Markus Gross. The blue-c distributed scene graph. In *Proc. of Eurographics Workshop on Virtual environments - EGVE*, pages 125–133, 2003.

[OBL07] Jan Ohlenburg, Wolfgang Broll, and Irma Lindt. "DEVAL - A Device Abstraction Layer for VR/AR". In *Universal Acess in Human Computer Interaction*, volume 4554 of *Lecture Notes in Computer Science*, pages 497–506. Springer, 2007.

[OC05] M. Ortega and S. Coquillart. "Prop-Based Haptic Interaction with Co-Location and Immersion: An Automotive Application". In *Proc. of the Int. Workshop on Haptic Audio Visual Environments and their Applications*, pages 23–28, Oct. 2005.

[OF04] Alex Olwal and Steven Feiner. Unit: modular development of distributed interaction techniques for highly interactive user interfaces. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '04, pages 131–138, New York, NY, USA, 2004. ACM.

[OHL+04] Jan Ohlenburg, Iris Herbst, Irma Lindt, Thorsten Fröhlich, and Wolfgang Broll. The morgan framework: enabling dynamic multi-user ar and vr projects. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '04, pages 166–169, New York, NY, USA, 2004. ACM.

[PBF02] Márcio S. Pinho, Doug A. Bowman, and Carla M.D.S. Freitas. Cooperative Object Manipulation in Immersive Virtual Environments: Framework and Techniques. In *Proceedings of VRST'02*, pages 171–178. ACM, 2002.

[PBWI96] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 79–80, New York, NY, USA, 1996. ACM Press.

[Phy] NVIDIA PhysX website. `http://developer.nvidia.com/object/physx.html`.

[PLI06] Amanda Parkes, Vincent LeClerc, and Hiroshi Ishii. Glume: Exploring Materiality in a Soft Augmented Modular Modeling System. In *Proc. of CHI*, pages 1211–1216, 2006.

[Pot96] Mike Potel. MVP: Model-View-Presenter — The Taligent Programming Model for C++ and Java. http://www.wildcrest.com/Potel/Portfolio/mvp.pdf, 1996.

[PVM] The PVM website. http://www.csm.ornl.gov/pvm/. Accessed August 23, 2012.

[PWBI98] Ivan Poupyrev, Suzanne Weghorst, Mark Billinghurst, and Tadao Ichikawa. Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques. *Computer Graphics Forum*, 17(3), 1998.

[Raz05] Sharif Razzaque. *"Redirected Walking"*. PhD thesis, University of North Carolina at Chapel Hill, 2005.

[Ree79] Trygve Reenskaug. The original MVC reports. http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf, 1979.

[RH92] Warren Robinett and Richard Holloway. "Implementation of Flying, Scaling and Grabbing in Virtual Worlds". In *Proc. of the Symp. on Interactive 3D Graphics*, pages 189–192, 1992.

[RHM+98] Patrick Reignier, Fabrice Harrouet, Serge Morvan, Jacques Tisseau, and Thierry Duval. Arévi : A virtual reality multiagent platform. In *Proceedings of the First International Conference on Virtual Worlds (VW'98), Paris, Lecture Notes in Computer Science, Artifial Intelligence series (LNCS/AI 1434)*, pages 229–240, juillet 1998.

[RHWF06] Kai Riege, Thorsten Holtkamper, Gerold Wesche, and Bernd Frohlich. The Bent Pick Ray: An Extended Pointing Technique for Multi-User Interaction. In *Proceedings of 3DUI'06*, pages 62–65. IEEE, 2006.

[RRS98] D. J. Roberts, M. D. Ryan, and P. M. Sharkey. "Combining Two Techniques for Overcoming Network Delays in a Distributed Virtual Ball Game". In *Proc. of the 2nd Workshop on Systems Aspects of Sharing a Virtual Reality at the ACM conf. on Collaborative Virtual Environments (CVE 98)*, 1998.

[RS97] D. J. Roberts and P. M. Sharkey. "Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays". In *Proc. of the IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 161–166, Jun 1997.

[RS01] Gerhard Reitmayr and Dieter Schmalstieg. An open software architecture for virtual reality interaction. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, pages 47–54, New York, NY, USA, 2001. ACM.

[RSJ02] Roy A. Ruddle, Justin C. D. Savage, and Dylan M. Jones. Symmetric and Asymmetric Action Integration during Cooperative Object Manipulation in Virtual Environments. *ACM Transactions on Computer-Human Interaction*, 9(4):285–308, 2002.

[RWOS03] David Roberts, Robin Wolff, Oliver Otto, and Anthony Steed. Constructing a Gazebo: Supporting Teamwork in a Tightly Coupled, Distributed Task in Virtual Reality. *Presence: Teleoperation and Virtual Environments*, 12(6):644–657, 2003.

[SC92] P. S Strauss and R. Carey. An object-oriented 3d graphics toolkit. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 341–349. ACM, ACM, 1992.

[SCF97] Maher Suleiman, Michèle Cart, and Jean Ferrié. Serialization of concurrent operations in a distributed collaborative environment. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge*, GROUP '97, pages 435–445, New York, NY, USA, 1997. ACM.

[SD99] Henry A. Sowizral and Michael F. Deering. "The Java 3D API and Virtual Reality". *IEEE Computer Graphics and Applications*, 19(3):12–15, 1999.

[SG93] C. Shaw and M. Green. "The MR Toolkit Peers Package and Experiment". In *IEEE Virtual Reality Annual International Symposium (VRAIS 93)*, pages 463–469. IEEE, 1993.

[SH02] D. Schmalstieg and G. Hesina. Distributed applications for collaborative augmented reality. In *Virtual Reality, 2002. Proceedings. IEEE*, pages 59 –66, 2002.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, 1985.

[SJF09] H. Salzmann, J. Jacobs, and B. Froehlich. Collaborative interaction in co-located two-user scenarios. In *Proc. of JVRC 2009 (Joint Virtual Reality Conference of EGVE - ICAT - EuroVR)*, pages 85–92, 2009.

[SLMA06] J. Sreng, A. Lecuyer, C. Megard, and C. Andriot. Using visual cues of contact to improve interactive manipulation of virtual objects in industrial assembly/maintenance simulations. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1013 –1020, sept.-oct. 2006.

[SRH03] Dieter Schmalstieg, Gerhard Reitmayr, and Gerd Hesina. Distributed applications for collaborative three-dimensional workspaces. *Presence: Teleoperators and Virtual Environments*, 12(1):53–68, 2003.

[SRH05a] Frank Steinicke, Timo Ropinski, and Klaus Hinrichs. "A Generic Virtual Reality Software System's Architecture and Application". In *Proc. of the International Conference on Augmented Tele-existence*, pages 220–227, 2005.

[SRH05b] Frank Steinicke, Timo Ropinski, and Klaus Hinrichs. A generic virtual reality software system's architecture and application. In *Proceedings of the 2005 international conference on Augmented tele-existence*, ICAT '05, pages 220–227. ACM, 2005.

[SSO94] Gerda J. F. Smets, Pieter Jan Stappers, and Kees Overbeeke. Designing in virtual reality: implementing perception-action coupling with affordances. In *Proceedings of the conference on Virtual reality software and technology*, VRST '94, pages 97–110, River Edge, NJ, USA, 1994. World Scientific Publishing Co., Inc.

[SSP+95] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng. "BrickNet: sharing object behaviors on the Net". *IEEE Virtual Reality Annual International Symposium (VRAIS 95)*, pages 19–25, Mar 1995.

[Ste08] Anthony Steed. "Some Useful Abstractions for Re-Usable Virtual Environment Platforms". In *Software Engineering and Architectures for Realtime Interactive Systems - SEARIS*, 2008.

[STP08] Ross T. Smith, Bruce H. Thomas, and Wayne Piekarski. Digital Foam Interaction Techniques for 3D Modeling. In *Proc. of VRST*, pages 61–68, 2008.

[SVP07] Pedro Sequeira, Marco Vala, and Ana Paiva. What can i do with this?: finding possible interactions between characters and objects. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 5:1–5:7, New York, NY, USA, 2007. ACM.

[SW94] Snowdon, D. and West, A. The AVIARY VR System: A Prototype Implementation. *6th ERCIM Workshop*, 1994.

[Tra99] Henrik Tramberend. Avocado: A distributed virtual reality framework. In *Proceedings of the IEEE Virtual Reality*, VR '99, pages 14–, Washington, DC, USA, 1999. IEEE Computer Society.

[UI00] Brygg Ullmer and Hiroshi Ishii. Emerging Frameworks for Tangible User Interfaces. *IBM Systems Journal*, 39(3-4):915–931, 2000.

[UIM92] "" UIMS 1992. A metamodel for the runtime architecture of an interactive system: the uims tool developers workshop. *SIGCHI Bull.*, 24(1):32–37, 1992.

[VGB99] Vaghi, I., Greenhalgh, C., and Benford, S. Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 42–49, 1999.

[WAB+97] R. Waters, D. Anderson, J. Barrus, D. Brogan, S. Mckeown, T. Nitta, I. Sterns, and W. Yerazunis. "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability". *Presence: Teleoperators and Virtual Environments*, 6(4):461–480, 1997.

[WHH+93] West, A., Howard, T., Hubbold, R., Murta, A., Snowdon, D., and Butler, D. AVIARY - A Generic Virtual Reality Interface for Real Applications. *Virtual Reality Systems (sponsored by the BCS), May 1992*, pages 213–236, 1993.

[WIA+04] Ryoichi Watanabe, Yuichi Itoh, Masatsugu Asai, Yoshifumi Kitamura, Fumio Kishino, and Hideo Kikuchi. The Soul of ActiveCube — Implementing a Flexible, Multimodal, Three-Dimensional Spatial Tangible Interface. *Computers in Entertainment*, 2(4):15–15, 2004.

[WL97] Colin Ware and Kathy Lowther. Selection Using a One-eyed Cursor in a Fish Tank VR Environment. *ACM Trans. Comput.-Hum. Interact.*, 4(4):309–322, 1997.

[WL10] Chadwick A. Wingrave and Joseph J. LaViola. "Reflecting on the Design and Implementation Issues of Virtual Environments". *Presence: Teleoper. Virtual Environ.*, 19(2):179–195, 2010.

[WNR+07] Betsy Williams, Gayathri Narasimham, Bjoern Rump, Timothy P. McNamara, Thomas H. Carr, John Rieser, and Bobby Bodenheimer. "Exploring Large Virtual Environments with an HMD When Physical Space is Limited". In *Proc. of the 4th Symp. on Applied perception in graphics and visualization*, pages 41–48, 2007.

[WR99] Colin Ware and Jeff Rose. Rotating virtual objects with real handles. *ACM Transactions on Computer-Human Interaction*, 6(2):162–180, 1999.

[YTAK95] M. Yoshida, Y. A. Tijerino, S. Abe, and F. Kishino. "A virtual space teleconferencing system that supports intuitive interaction for creative and cooperative work". In *SI3D'95: Proceedings of the symposium on Interactive 3D graphics*, pages 115–122, New York, NY, USA, 1995. ACM.

[Zam05] Chadi Zammar. *"Interactions coopératives 3D distantes en environnements virtuels : gestion des problèmes réseau"*. PhD thesis, IRISA/INSA de Rennes, 2005.

[ZBM94] Shumin Zhai, William Buxton, and Paul Milgram. The "Silk Cursor": Investigating Transparency for 3D Target Acquisition. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 459–464, New York, NY, USA, 1994. ACM Press.

[ZD03] Chadi Zammar and Thierry Duval. Management and Awareness of Delay Perception when Interacting within Networked Virtual Environments. In *VRIC 2003*, Laval, France, May 2003.

[ZF05] Xiaolong Zhang and George W. Furnas. "mCVEs: Using Cross-Scale Collaboration to Support User Interaction with Multiscale Structures". *Presence: Teleop. & Virtual Env.*, 14(1):31–46, 2005.

[ZJ98] Pavel Zahorik and Rick L. Jenison. Presence as being-in-the-world. *Presence: Teleoper. Virtual Environ.*, 7(1):78–89, February 1998.

Abstract

This document aims at providing some cues in order to address the essential requirements about the design of 3D Collaborative Virtual Environments (CVE).

We have identified six essential topics that must be addressed when designing a CVE. For each of them, we present a state of the art about the solutions that can address this topic, then we show our own contributions: how we improve existing solutions and what are our new propositions.

Choosing a model for the distribution of a CVE

We need a distribution model to distribute as efficiently as possible the content of a CVE among all the nodes involved in its execution, including the machines of the distant users. Our proposition is to allow CVE designers to mix in a same CVE the three main distribution models usually encountered: centralized on a server, totally replicated on each site, or distributed according to a hybrid distribution model.

Choosing a model for the synchronization of these nodes

To maintain consistency between all the nodes involved in the execution of a CVE, we must choose between a strong synchronization or a relaxed one, or an in-between solution. Our proposition is to manage some temporary relaxation of the synchronization due to network breakdowns, with several synchronization groups of users, making them aware of these network breakdowns, and to allow some shared objects to migrate from one site to another.

Adapting the Virtual Environment to various hardware systems

VR applications must be adapted to the software and to the hardware input and output devices that are available at run-time, in order to be able to deploy a CVE onto different kinds of hardware and software. Our solution is the PAC-C3D software architectural model which is able to deal with the three main distribution modes encountered in CVE.

Designing interaction and collaboration in the VE

Expressing the interactive and collaborative capabilities of the content of a CVE goes one step beyond geometric modeling, by adding interactive and collaborative features to virtual objects. We propose a unified model of dialog between interactive objects and interaction tools, with an extension to Collada in order to describe interactive and collaborative properties of these interactive objects and interaction tools.

Choosing the best metaphors for collaborative interactions

Most of the time single-user interaction tools and metaphors are not adapted to offer efficient collaboration between users of a CVE. We adapt some of these tools and metaphors to collaborative interactions, and we propose new really collaborative metaphors to enhance real multi-user collaborative interactions, with dedicated collaborative feedback.

Embedding the users' physical workspaces within the CVE

Taking into account users' physical workspaces makes it possible to adapt a CVE to the hardware input and output devices of the users, and to make them aware of their physical limitations and of those of the other users, for better interaction and collaboration. We propose the Immersive Interactive Virtual Cabin (IIVC) concept to embed such 3D representations in CVE.